Local Rotational Symmetries

by

Margaret Morrison Fleck

B.A., Linguistics, Yale University
(1982)

Submitted to the Department of Electrical Engineering
and Computer Science in partial fulfillment of the
requirements of the degree of

Master of Science in
Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

September 1985

Local Rotational Symmetries

by

Margaret Morrison Fleck

Submitted to the Department of Electrical Engineering
and Computer Science in partial fulfillment of the
requirements of the degree of Master of Science
in Electrical Engineering and Computer Science

Abstract:

This thesis describes a new representation for two-dimensional round regions called Local Rotational Symmetries. Local Rotational Symmetries are intended as a companion to Brady's Smoothed Local Symmetry Representation for elongated shapes. An algorithm for computing Local Rotational Symmetry representations at multiple scales of resolution has been implemented and results of this implementation are presented. These results suggest that Local Rotational Symmetries provide a more robustly computable and perceptually accurate description of round regions than previous proposed representations.

In the course of developing this representation, it has been necessary to modify the way both Smoothed Local Symmetries and Local Rotational Symmetries are computed. First, grey-scale image smoothing proves to be better than boundary smoothing for creating representations at multiple scales of resolution, because it is more robust and it allows qualitative changes in representation between scales. Secondly, it is proposed that shape representations at different scales of resolution be explicitly related, so that information can be passed between scales and computation at each scale can be kept local. Such a model for multi-scale computation is desirable both to allow efficient computation and to accurately model human perceptions.

Thesis Supervisor: Dr. J. Michael Brady
Title: Senior Research Scientist

Acknowledgements

I could never have finished this work without the comfort, aid, and inspiration of a large number of people. First, and foremost, I would like to thank my advisor, Mike Brady. Without his intellectual guidance and patient encouragement, this thesis would never have been written. His constant enthusiasm for research makes him a pleasure to work with.

I owe an equal, but more diffuse debt of gratitude to my friends around the AI Lab, particular the members of the Shape Group: Phil Agre, Steve Bagley, Dave Braunegg, Jon Connell, Scott Heide, Jon Kaplan, Jean Ponce, Eric Saund, Alan Yuille. You've found the flaws in my theories and given me ideas for how to fix them. You've been supportive when research or life got discouraging. Many of the ideas we all use are common property whose exact authorship has been blurred by long conversations and much beer.

Jon Connell and Alan Yuille are to be especially thanked for reading drafts of this thesis and finding bugs that I would never have been able to find myself.

I would also like to thank Chris Lindblad and Scott Jones for bringing up the LN01 printer and thus allowing me to get high-quality hardcopy of my grey-scale images and shape analyses.

Finally, I would like to thank my parents, who have always been there when I needed them.

The research described in this thesis was done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's Artificial Intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-80-C-0505, the Office of Naval Research under contract number N00014-77-C-0389, and the System Development Foundation. The author is also supported by the Fannie and John Hertz Foundation and by the Bell Laboratories Graduate Research Program for Women.

Table of Contents

Chapter 1: Introduction

The goal of this thesis is to develop a computer model of how people represent shape. People analyze sensory data, including visual images, into representations of 2-dimensional and 3-dimensional shapes. These representations can then be used for guiding motion planning, for practical reasoning about objects and actions, and for representing the meanings of natural language terms that refer to objects or to aspects of the shape of objects. These representations are computed extremely quickly and robustly. Most existing systems for representing shape are a relatively poor match to human capabilities for representing shape: they cannot be robustly computed for natural shapes, nor do they produce analyses of shapes which match human judgements.

The starting point of this work is the Smoothed Local Symmetry representation described by Brady (1983) and Brady and Asada (1984). A Smoothed Local Symmetry representation of a 2-dimensional shape picks out the axes of elongated regions in the shape and produces descriptions of these regions. For example, consider the grey-scale camera images of familiar objects shown in Figure 1 (top). Smoothed Local Symmetry representations are computed from the boundaries of regions, rather than directly from the grey-scale image. Thus, in order to analyze the shapes of these objects, we first extract the boundaries of regions in the image, as shown in Figure 1 (bottom). Figure 2 shows the axes found by a Smoothed Local Symmetry analysis of these figures. This analysis creates intuitively reasonable representations for elongated or pointed regions such as the handles of the spanner wrench and the teaspoon, the pointed jaws of the wrench, and the main axes of the gourd, the pear, the squash, and the bowl of the teaspoon. The figure shows the axes of these regions, as well as other smaller axes that will be removed in later analysis (see Connell 1985).

Smoothed Local Symmetry representations of an object can be computed at coarser and finer degrees of resolution, thus capturing both the overall shape of an object and fine details such as edge texture. Coarse-resolution versions of a grey-scale image are obtained by repeatedly smoothing the image with a Gaussian and sampling the result. The boundaries are then extracted from each of these progressively smoothed versions of the original image.[1] For example, Figure 3 shows the original and smoothed versions of the lemon image, as well as the

[1] This is a change from what is described in Brady and Asada (1984) which will be explained in more detail later.

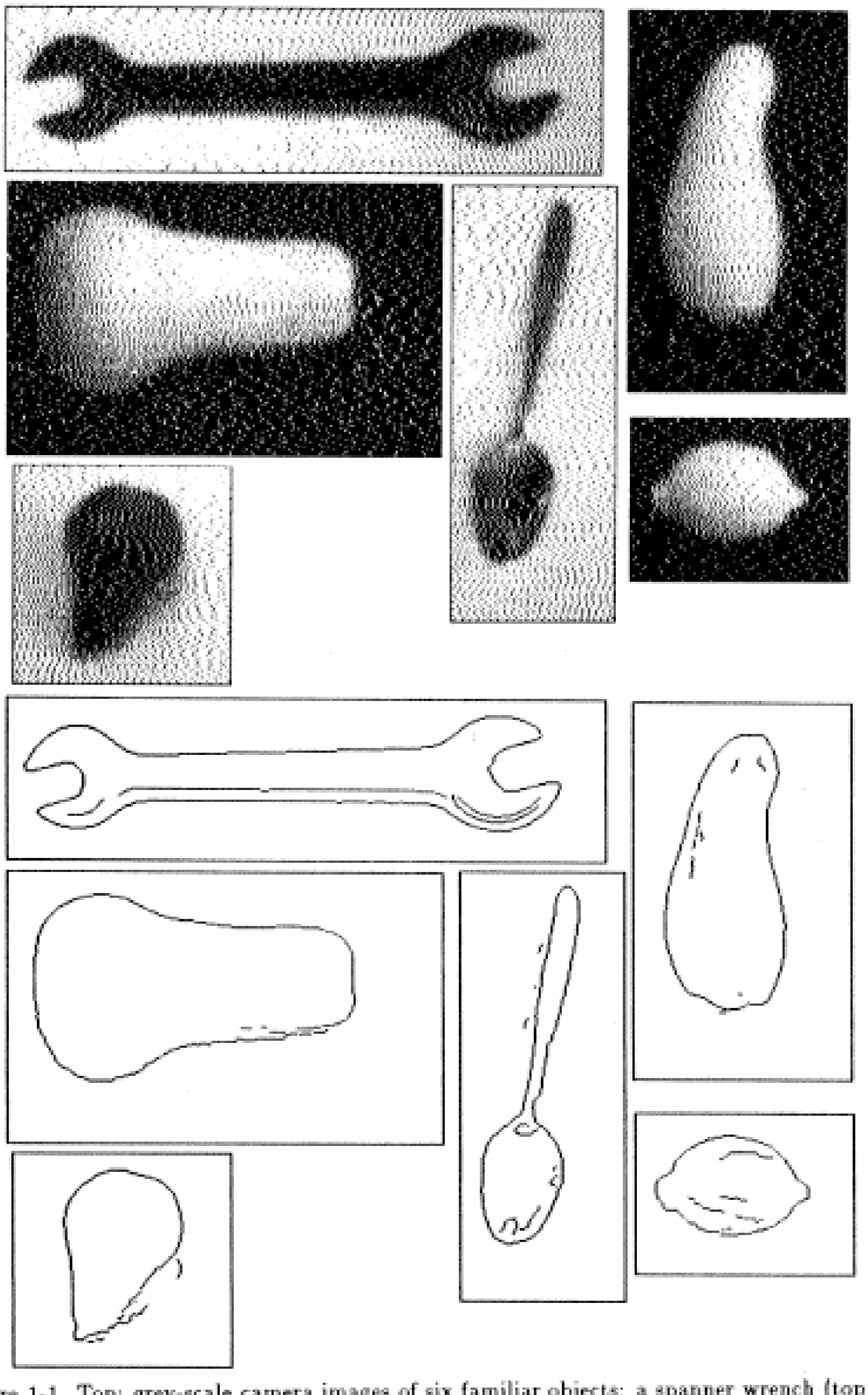

Figure 1-1. Top: grey-scale camera images of six familiar objects: a spanner wrench (top left), a gourd (middle left), a pear (bottom left), a teaspoon (middle), a squash (top right), and a lemon (bottom right). Bottom: Boundaries of regions in these grey-scale images. These boundaries are locations of sharp changes in intensity found by the edge finder described by Canny (1983).

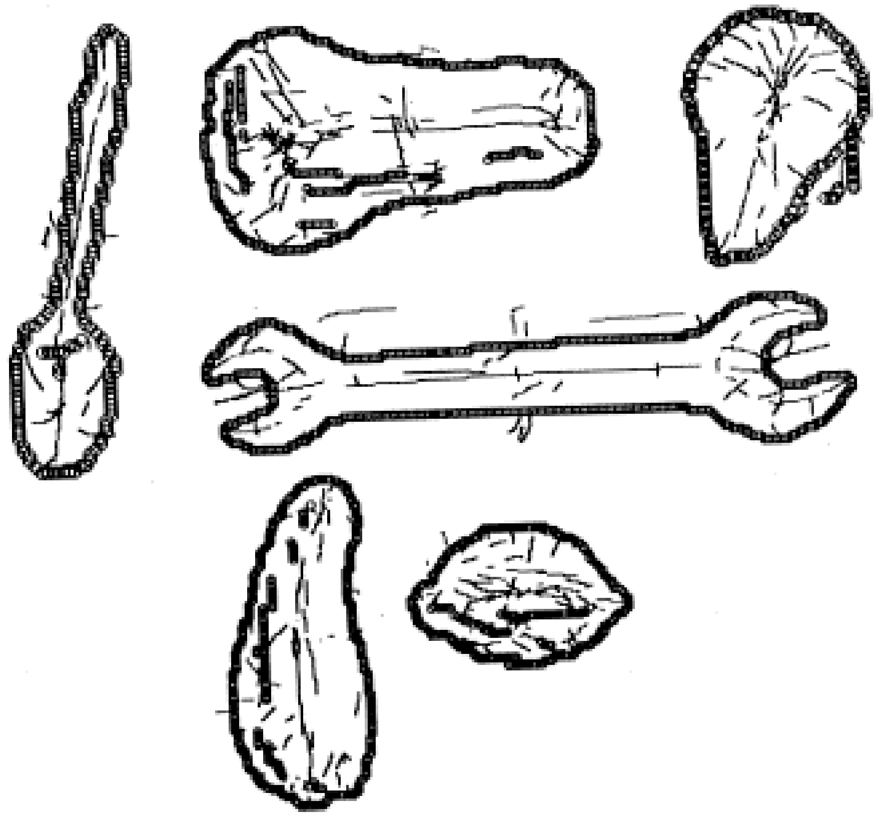

Figure 1-2. A Smoothed Local Symmetry analysis of the images from Figure 1-1. The thick lines are the region boundaries from the images. The thin lines are the axes of Smoothed Local Symmetry regions. Smoothed local symmetry regions include elongated regions such as the handle of the spanner wrench and the main axes of the gourd, squash, and teaspoon, and also pointed regions such as the end of the pear and the jaws of the spanner wrench.

boundaries obtained from these images.[2] For a shape with fine-scale texture, the fine-scale and coarse-scale representations may be qualitatively different. For example, Figure 4 shows an image of an oak leaf, the region boundaries in the image, and a Smoothed Local Symmetry analysis of the leaf at two scales of resolution, one finer and one coarser. The fine-scale analysis picks up the axes of the lobes of the leaf and the coarse-scale analysis picks up the main axis of the leaf, which is obscured by details at the fine scale.

However, the Smoothed Local Symmetry representation does not provide intuitively acceptable analyses for round regions, such as the lemon and the round ends of the gourd, pear, and squash in Figure 1. Furthermore, although it pro-

[2] In fact, the Smoothed Local Symmetry analyses shown in Figure 2 use boundaries from slightly smoothed versions of the images, rather than from the original fine-scale image.

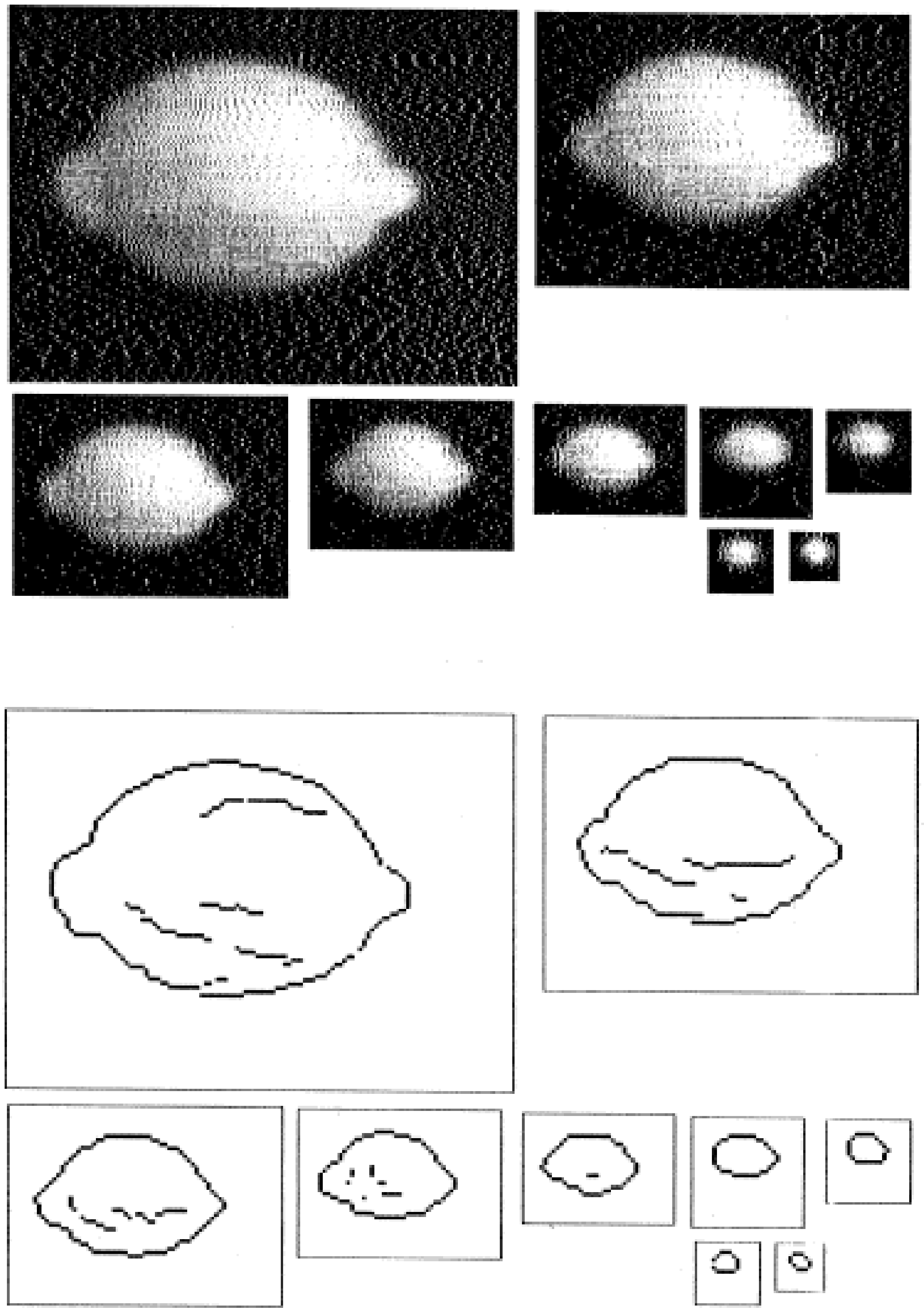

Figure 1-3. Top: The grey-scale image of the lemon, repeatedly smoothed with a Gaussian and sampled. Bottom: Boundaries of regions extracted from the smoothed images.

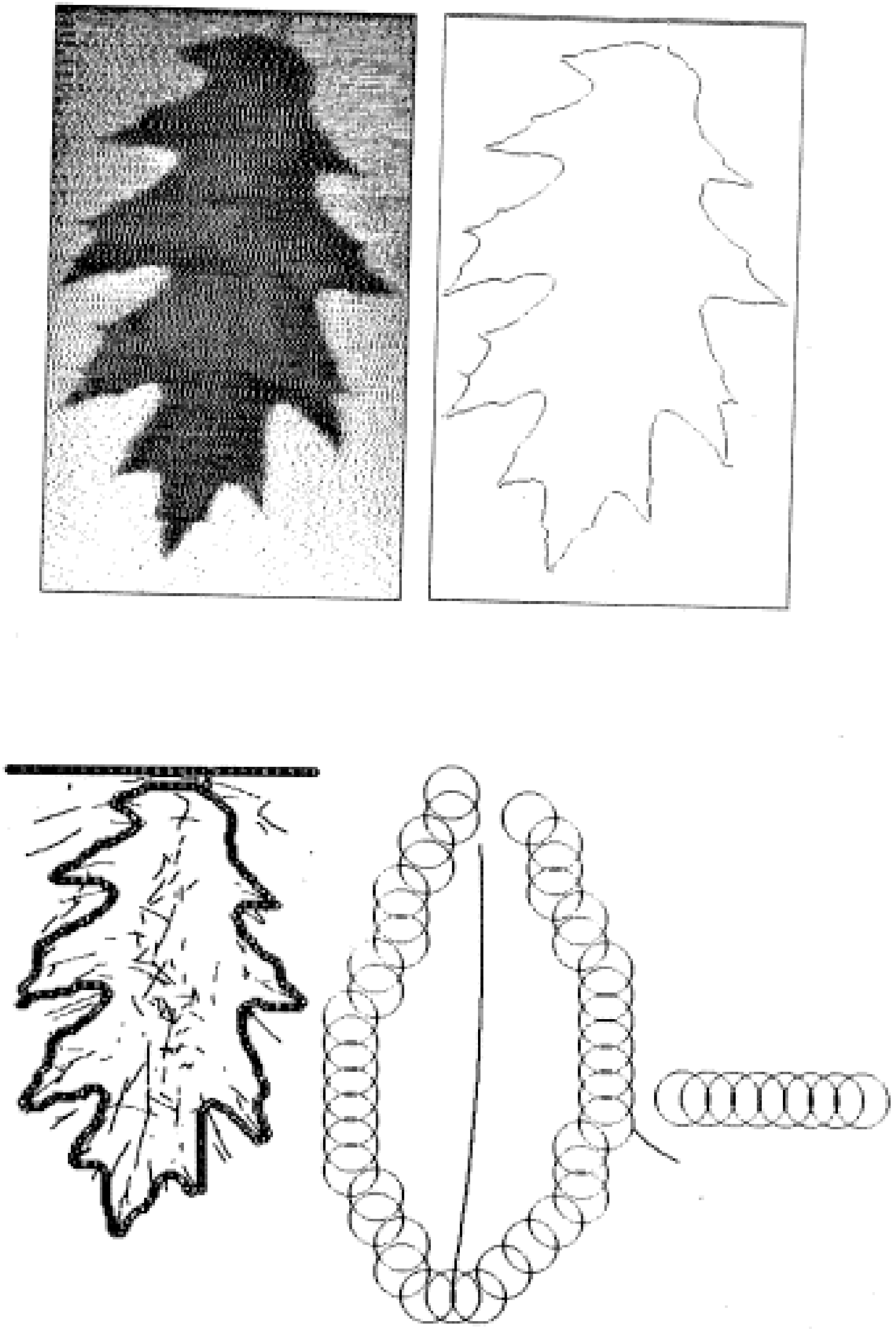

Figure 1-4. Top: a grey-scale image of an oak leaf and region boundaries extracted from it by the edge finder. Bottom: Smoothed Local Symmetry analysis of the image at a fine scale and at a coarse scale. The fine-scale analysis finds the axes of the lobes of the leaf. The coarse-scale analysis finds the main axis of the leaf, which was obscured by detail in the finer-scale image.

vides an interpretation of the ends of the wrench and the bowl of the teaspoon in terms of an axis, it does not capture the fact that these regions can also be described as round, without a distinguished axis. Furthermore, the Smoothed Local Symmetry representation is unstable on such regions. Therefore, I propose a companion representation, *Local Rotational Symmetries*, to represent round regions in 2-dimensional images. Later in this thesis, I will discuss detailed criteria for a good representation of round regions and argue that Local Rotational Symmetries are more robustly computable than other representations for round regions and more closely model human perceptions of shape.

I have implemented an algorithm for computing Local Rotational Symmetry representations. Figure 5 shows the regions that found by the program for the images in Figure 1. The program identifies the bowl of the teaspoon, the body and round tips of the lemon, and the round ends of the spanner, the pear, the gourd, and the squash, as well as a few round regions of spectral reflection on the spoon. The program also finds the squarish cut-outs of the wrench and the squarish end of the gourd, all of which are also more round than elongated. These round regions can also be computed at a variety of scales of resolution. For example, Figure 6 shows a grey-scale image of a cog, the region boundaries in the image, and the Local Rotational Symmetry regions computed for this image at a fine scale of resolution and at a coarser scale. The coarse-scale analysis represents the overall shape of the cog as one round region (with a small round region in the center). The fine-scale analysis finds the teeth of the cog and the small half-round regions between them.

In the course of developing this representation for round regions, I have had to re-think various aspects of the design of local symmetry representations for shape. My implementation of Local Rotational Symmetries uses image smoothing, rather than boundary smoothing to produce representations of an image at multiple scales. This change in the smoothing method allows qualitative changes in representation between scales and makes analysis of natural images more robust. Furthermore, unlike current implementations of Smoothed Local Symmetries, analyses at different scales of resolution are explicitly related. Information from coarser scales can be used to guide analysis at finer scales, allowing the algorithm for computing symmetries to be strictly local. Further, the explicit relationships between scales could be used to relate symbolic representations at adjacent scales and perhaps produce a representation of regions in an image across scales

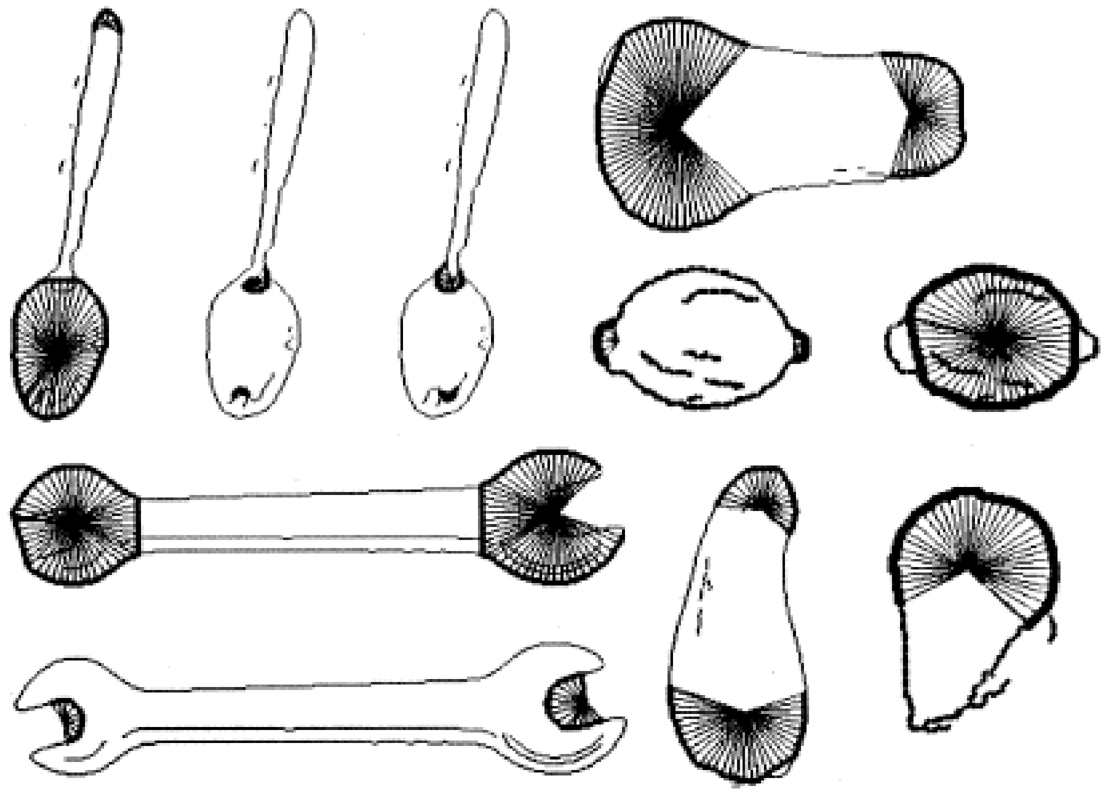

Figure 1-5. Local Rotational Symmetry analysis of the images in Figure 1. The thin lines show the region boundaries found in the finest-scale version of these images. The boundary of each round region is shown as a thicker line and selected radii from the center of the region to the boundary are also displayed. For example, four regions are found for the teaspoon: the bowl, the round end of the handle, and four small round regions in edges from spectral reflections on the bowl.

analogous to the scale-space analysis of inflections in a one-dimensional signal proposed by Witkin (1983). These ideas could also be applied to a re-implementation of Smoothed Local Symmetries.

A roadmap for the rest of the thesis is as follows:

- In Chapter 2, I discuss what properties a shape representation should have in order to accurately model human perceptions.
- In Chapter 3, I review the definition of a Smoothed Local Symmetry, discuss problems with using Smoothed Local Symmetries to describe round regions, and present the definition of a Local Rotational Symmetry. In addition to the local geometry of symmetry regions, I discuss how to evaluate regions, choose the perceptually best regions to describe a shape, and join disconnected pieces

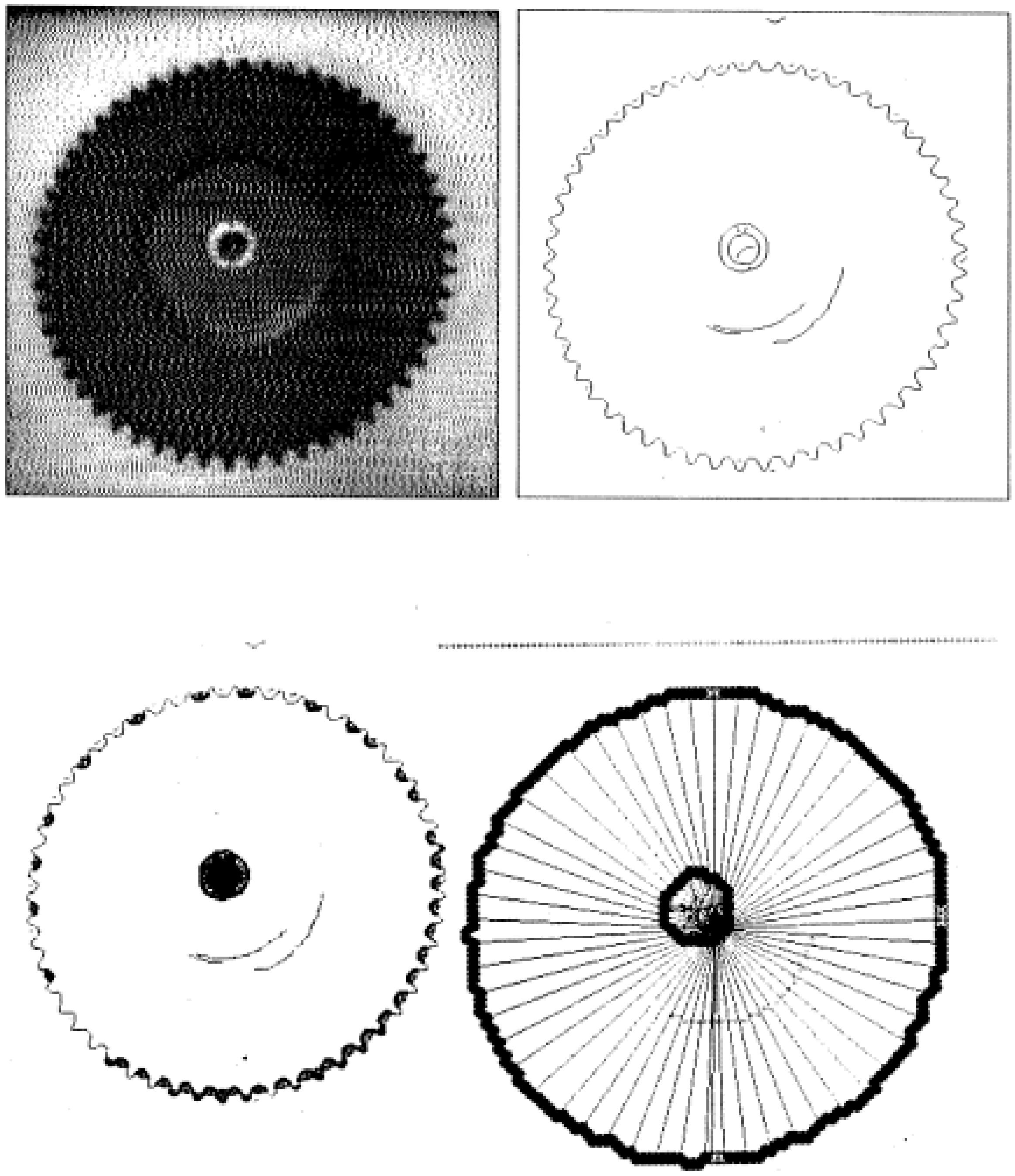

Figure 1-6. Top: a grey-scale image of a cog and the region boundaries in the image. Bottom: Local Rotational Symmetry regions computed for the image at a fine scale and at a coarser scale of resolution. The coarse-scale analysis represents the overall shape of the cog as one round region, whereas the fine-scale analysis finds the small half-round regions in the teeth of the cog. Only some of the regions in the teeth are located, because the teeth are near the limit of resolution of the current implementation. The teeth toward the lower righthand edge of the cog are slightly larger and so more of them have been found.

of boundary to form connected regions.

- In Chapter 4, I discuss details of the implemented multi-scale algorithm for computing Local Rotational Symmetries and presents results of that algorithm. This chapter contains more detailled descriptions of the algorithms used to compute LRS regions, as well as examples of program output (Section 4.7).
- In Chapter 5, I compare local symmetry representations to other shape representations and discuss issues in representing images at a fixed scale of resolution. Important issues discussed include: how to build high-level shape descriptions from the raw symmetry regions, why the representational system should allow multiple representations for some shapes in order to be stable and accurately model human perceptions, and whether there is a constraint that perceptually salient regions have uniform or slowly changing color.
- In Chapter 6, I discuss issues in representing an image at multiple levels of resolution, comparing the method I use to previous use of multiple-scale representations. Important issues include: alternative ways to create representations at multiple scales of resolution, how to relate representations at different scales, why exhaustive computation at each scale should be local.
- Chapter 7 is a summary and conclusion.
- The Appendix contains a listing of the LISP code used in computing Local Rotational Symmetries.

Chapter 2: Criteria for a good shape representation

The goal of this research is to model the way that humans represent shape. When a person looks at the world, he processes the raw visual and other sensory input extremely quickly into a form which allows him to reason about how to interact with the world. Representations of objects are used for:

- Interpreting a scene as a set of objects, tracking the positions and form of these objects over time, and planning motion, hand movements, eye movements involving these objects;
- Identification and description of objects and scenes using natural language;
- Practical reasoning about processes and the behavior of objects in the world.

By studying human behavior that is dependent on representations of objects and scenes, we can infer properties of these representations.

I am being deliberately careful to emphasize that evidence about how humans represent shape can only come from behavior dependent on these representations. Modelling human behavior is different from building an algorithm to accomplish some technically defined task, such as accurately determining the orientation of points on an object from the shading of the object under some particular lighting conditions, determining the 3-dimensional shape of an object from one or more 2-dimensional views, representing objects in terms of a set of mathematically convenient primitives, e.g. ellipses. While algorithms for doing such tasks well may be interesting in and of themselves and may serve as inspiration in building models of human perception, they are not a solution to the problem of modelling how humans represent the world. In modelling human behavior, one must determine not only how to do the task well, but also exactly what task humans actually do. A good example of the distinction is to consider the terms "ellipse" and "oval". The mathematical definition of "ellipse" refers to a very specific class of 2-dimensional shapes. This class of shapes is related to, but distinct from, the class of shapes that the (informal) English words "oval" and "ellipse" refer to. While people apply the English words consistently to naturally occuring shapes, it is less clear how well they can identify instances of shapes meeting the mathematical definition.

In using natural language and in doing practical reasoning about the world, people divide the world up into objects and sub-objects and regions of empty space. Objects are grouped into so-called natural kinds, classes, or types of objects, such as hammers, leaves, serrated leaves, pears, corners, round objects. The simplest hypothesis is that the structure of this class system reflects the structure of the representational system. That is, objects classified as the same type of object are represented similarly. How people classify objects can be determined by studying how natural language words are applied, observing how people do practical reasoning, and by directly asking people to classify objects. Factors which seem to be important in classifying objects include shape, color, functional properties such as weight and flexibility, tactile properties such as smoothness, and so forth.

This thesis will be exclusively concerned with representing shape, primarily 2-dimensional shape. Shape is a good place to start investigating object representations, because many objects can be recognized by their shape alone, without color or other functional information. I am working with 2-dimensional shapes for several reasons. First, it is possible to recognize many 3-dimensional objects from 2-dimensional views or from line drawings without shading, stereo or other direct evidence of 3-dimensional shape. This suggests that people have 2-dimensional or augmented 2-dimensional representations of visual images which they can relate to 3-dimensional shape models. (The 2 1/2-D Sketch described in Marr (1982) is an example of an augmented 2-dimensional representation.) Secondly, people can recognize and describe purely 2-dimensional shapes, such as written letters, squares and triangles, or the shapes of flat objects such as tree leaves. Finally, it is likely that representational techniques used for describing 2-dimensional shape perception can be extended to the 3-dimensional case. The objects I use as examples were chosen because they have a distinctive shape which can be well-represented with one 2-dimensional black and white picture, e.g. flat objects such as leaves or spanner wrenches. For some of the examples shown, this restriction means that the reader may need some context to identify the object. For example, it is difficult or impossible to identify vegetables and fruits out of context without color and size information. Information about 3-dimensional shape would also help disambiguate certain of the shapes.

In order to determine what human representations of shape are like, we need to look at how classification of objects is affected by various types of changes to objects, how people describe objects, what types of objects they judge similar, and so forth. Human judgements and behavior allow us to infer that a shape representation should have the following properties if it is to accurately model human shape representations:

- It should be invariant under simple transformations (translation, changes in size, rotation);
- It should be stable under noise;
- It should allow arbitrary amounts of detail to be computed and also allow abstraction from details;
- It should be fast and robust;
- Its judgements of relative similarity, its judgements of relative complexity, and its descriptions of differences between two objects should match answers humans give;
- It should make explicit concepts such as "axes" and "centers" which occur in human descriptions of shapes;
- It should represent complex objects in terms of sub-parts and the descriptions of parts should be stable under attachment;
- It should be stable under change of 3-D viewpoint.

Similar criteria are an obligatory preface to most work on shape representation. The paper to which I owe the greatest intellectual debt is Marr and Nishihara (1978), although the way I divide up the problem is not quite compatible with theirs. These criteria will be discussed in detail in the rest of this chapter.

I should note that this point that detailed psychological and linguistic work would be needed to determine the fine details of human judgements and behavior. In later chapters, I will point out places where such evidence might be useful in refining the theories presented in this thesis. Zusne (1970) gives a survey of relevant psychological data. However, I will not present much detailed formal data, because there is not much formal data that bears on the issues considered in this thesis. There are several reasons behind the lack of formal data:

- Most existing formal systems for representing shape do a poor job of modelling human capabilities. They cannot, for example, reliably recognize common household objects such as scissors, hammers, and pears. Plausible theories can be distinguished from inadequate ones on the basis of coarse-scale facts about human perceptions obtained by informal observation of human behavior, linguistic data, and so forth.

- Much of the literature is concerned with tasks only tangentially related to representing the shape of 2-dimensional regions, e.g. studies of the perceived shape of subjective contours, texture perception, differences in perception of a shape as a function of location in the visual field, and perceptual grouping.
- Evidence from sources such as linguistic data often reflects factors other than 2-dimensional shape. Detailed formal analysis of such data will be difficult to do until we have developed a fuller theory that includes preliminary representations of 3-dimensional shape, color, function, and so forth.
- It is difficult to design meaningful psychological experiments or methods of analyzing linguistic data except with reference to a preliminary theory which is precise and already an approximate match to the facts. Existing methods for quantifying properties of shapes (see Zusne 1970, chapter 5) are crude and mostly limited to polygons or relatively simple shapes.
- Experiments that use more general or more perceptually plausible properties but without precisely defining them are difficult to interpret. For example, the work presented in Biederman (unpubl.) is difficult to relate to issues of processing real images because the theory of perception on which it is based is too vague.

The goal of this thesis is to develop a theory of shape representations which matches human perceptions sufficiently well that detailed experimental testing and refinement is appropriate.

2.1. Invariance under simple transformations

The size of an object and its orientation and location in the visual field should be represented independently from other features of the shape. This is needed for several reasons. First, human perceptions of shape are stable under small changes in size, orientation, and location. Secondly, although relative size, orientation, and location of objects within a scene or sub-objects within an object can be important in classification of the scene or object, absolute size, orientation, or location of an object in space or in the visual field is generally not important. Finally, natural language contains specific terms for describing size, orientation, and relative location. Words describing other aspects of shape are largely independent of changes in these parameters. All of these facts suggest that regions are represented by size, orientation, and location parameters, plus a shape description which is invariant under changes in these parameters.

Relative size, location, and orientation of sub-parts within an object or of objects within an arrangement can be extremely important perceptually. For example, the fact that the ends of the wings of an airplane are attached to its body is crucial in identifying it as an airplane. Although the size and orientation of the wings relative to the body can vary somewhat, they must be within a fixed range for the object as a whole to look like an airplane (cf. Brooks 1981 and Connell 1985). Similarly, in order for a set of people to be said to be "standing in a straight line", they must meet criteria on the location of each person relative to the other people.

Changes in the absolute size, location, and orientation of an object or set of objects are rarely important in classifying the object or scene. Changes can occur between different views of the same object, as one moves closer to an object or the object is rotated around the line of sight and this does not generally affect a human's identification of the object. Further, even from a constant viewpoint, two different objects of the same natural class (e.g. two objects that would be labelled with the same natural language word) may differ in size, location, and orientation. The main exception to this is that some recognition and labelling processes are sensitive to large differences in orientation. For example, squares and diamonds are distinguished by their orientation. Also, some objects and scenes are difficult to recognize when upside down. However, these exceptions are minor compared to the general pattern of separation of size, location, and orientation from shape properties.

2.2. Stability under noise

Human perception of object shape seems not to be affected much by noise or clutter in the visual image. There are a variety of sources of noise. First, there may be clutter or texture or color patterns in, on, or around the objects themselves. Secondly, there may be noise or other types of degradation in the visual image of the scene. This occurs most conspicuously in laboratory experiments on perception and in images transmitted by media such as television.

Human perception of shape also seems to be relatively robust in the presence of small imperfections in region boundaries. These include small gaps, two parallel boundary curves instead of one, and small spurs off the sides of boundaries. Such imperfections occur in output from the edge finder used in our analyses of images and could in that context be viewed as indicating problems with the edge finder. However, such imperfections also occur in line drawings of objects produced by people and do not cause drastic changes in the perception of shapes of regions in these drawings. Thus, whether or not the edge finder could be improved, shape representation algorithms must not be sensitive to such types of small imperfections in boundaries. Small imperfections in boundaries should be noted in detailed representations of a shape, but they should not be allowed to cause large disruptions in the overall representation.

2.3. Abstraction from detail

The representations humans create for objects or scenes seem to allow one to represent arbitrary amount of detail. There do not seem to be any classes of shapes that cannot be distinguished if sufficiently detailed processing is done. At the same time, the representations allow one to abstract away detail when it is unnecessary. There are a variety of types of detail that need to be abstracted away from:

- Features and noise much smaller than the overall shape of a region;
- Texture covering a region;
- Clutter of approximately the same size as the region, such as an internal color boundary dividing a region into two pieces.

When people abstract away detail, the abstracted representations may be *qualitatively* different from the detailed representations. Good examples of this occur in practical reasoning. Reasoning about the behavior of a complex object is often done by reasoning about an abstracted representation of the object. For example, a detailed representation of a hammock is that it is a regular mesh of cords. An abstracted representation of a hammock might be that it is a thin sheet. Using the flat sheet representation, people can reason about the behavior of a hammock under applied forces such as a person lying on it. Reasoning about the behavior of a hammock directly from the physics of the mesh representation is much more difficult: people who are quite familiar with a hammock and can easily predict its overall behavior may be unable to explain how the mesh succeeds in staying uniform and locally planar and why the mesh does not loosen up and create big holes.

When an object has both a relatively detailed representation and an abstracted representation, these representations seem to be related. That is, features in one representation are matched with the corresponding features (if any) in the other representation, so that facts deduced about one representation can be converted into facts about the other representation.

2.4. Speed of computation and robustness

Humans compute representations of visual images, including the objects in them, extremely quickly. The computation that is done involves some choice of how much detail is to be represented for each part of the scene, depending on the goals of the observer. Computation of representations of common objects in sufficient detail to recognize them, including choice of appropriate amounts of detail and and the recognition process itself, can be done extremely fast and robustly. Practical reasoning about common objects can also be done rapidly. This implies the following facts:

- In order for computation to be fast, data dependencies during the computation must be local. Otherwise computation of shape descriptions would slow down more than linearly on large or complex images.
- There cannot be more than a few alternative representations computed for any given shape. Otherwise, reasoning that uses these representations, including identification of objects, would slow down drastically on certain types of shapes.

I should note that I am not worrying here about small changes in processing speed of the sort that might take detailed experimentation to discover. Rather, I am worried about the several order of magnitude slowdown in processing circles relative to processing squares that would occur if circles were given as many representations as they have diameters but squares were given one representation. Or, to take another example, there would be a drastic slowdown on large and complex images if processing time grew much faster than linearly in the size and complexity of the input image. Such differences in processing time would be blatantly obvious to casual observation and do not seem to occur in human processing of visual input.

2.5. Relative similarity, complexity

The representation should be stable under deformations that seem "minor" to people or which seem to have minor effects on practical reasoning about objects. The relative similarity of the representations of two objects should match human judgements of their relative similarity. The differences between the representations of two objects should match human descriptions of their differences. One aspect of matching human similarity judgements is that there should not be sudden changes in representation in the middle of what humans judge to be a continuous smooth variation in shape (stability).

Evidence about human similarity judgments can be obtained by direct questioning or by examining what words are used to describe different types of objects (cf. Labov 1973). Another way to get evidence on human similarity judgements is to observe what types of objects people expect to behave similarly in practical reasoning about objects and actions. Types of objects that people expect to behave similarly should be represented similarly. For example, to a human observer, it is obvious that a hexagonal pencil could either roll and slide, depending on the circumstances. Rolling and sliding involve very different aspects of the shape of the pencil: rolling requires that the cross-section be "close" to circular and sliding requires that the pencil have a side that is "close" to flat. Thus, a pencil should have a shape representation which is similar both to a cylinder and to a flat-sided object, or else two representations capturing these distinct views of the pencil's shape.

In addition, relative complexity of the representations of two objects should match human judgements of their relative complexity. For example, suppose that the representational system measures complexity of shapes by counting the number of elongated and round regions in the representation and suppose that this is the correct measure of shape complexity for determining how long it takes to reason about objects of various shapes. In that case, circles could not have region representations explicitly computed for infinitely many or even a very large number of the possible axes of the circle. If circles were given a complicated representation involving large numbers of axes, whereas rectangles and ellipses have only one salient axis and an airplane only maybe 5-10, this would imply that circles are drastically more complicated perceptually than the other figures. That is, circles would seem more much complicated to humans and would take longer to reason about. This does not seem to be the case. If other measures of complexity are used, similar arguments apply, mutatis mutandis.

2.6. Axes, subparts, and other features of shapes

People use axes and widths to describe elongated shapes, centers and radii to describe round shapes. They seem to have clear intuitions as to where centers and axes lie. This implies that these features of shapes are psychologically real. People refer to a region and *its* boundary, explicitly recognizing both and relating them to each other. Thus, a model of how humans represent shape should make these notions explicit and its use of them should agree with human judgements.

People describe complex shapes in terms of subparts which are adjoined or cut out of each other. This implies that the representations should build complex shapes out of simpler ones and that the subparts postulated should agree with human judgements about subparts. The representations people use seem to be relatively stable under attachment of other objects to the object being described. For example, the handles of a knife, fork, and spoon from the same silverware pattern will be perceived as similar, despite the extremely different business ends attached to them. How much the representation of a region is disrupted by attached parts seems to be a function of how much of the boundary of the region is disrupted and which features of the boundary are destroyed.

2.7. Stability under change of 3-D view

The world we live in is 3-dimensional and some of our sensory data, e.g. tactile data, directly reflect 3-dimensional situations. However, the images received by the visual system are 2-dimensional, with partial information about surface orientation added by stereo matching and shape from shading. A given 3-dimensional scene can generate extremely variable 2-dimensional images, depending on the direction from which the scene is viewed. Changes to an image which depend on viewpoint include:

- smooth deformation of regions due to rotation;
- qualitative changes in the shape of objects due to rotation;
- occlusion, including self-occlusion.

A human's ability to recognize objects and to relate visual information to 3-dimensional models of objects seems to be able to compensate for some types of changes in view, particularly smooth deformations. Other types of changes make the object difficult to identify, particularly qualitative changes in shape, such as the difference between an end view and a side view of a bottle. In addition, the human visual system seems to be able to compensate for changes in the 3-dimensional scene itself, including differences in shadows and shading due to changes in lighting.

In this thesis, I will avoid the difficult questions concerning the relationships between 2-dimensional models or augmented 2-dimensional models (cf. Marr 1982) of visual scenes and 3-dimensional models of objects and scenes by only considering 2-dimensional shapes. The objects used in examples were chosen specifically so that the shape of the object could be well represented by a single 2-dimensional view. Some care was taken in photographing objects to avoid shadow and occlusion, although both effects do occur in our data.

Chapter 3: Local Symmetry Representations for Shape

This chapter reviews the Smoothed Local Symmetry representation for elongated regions and presents the new Local Rotational Symmetry representation for round regions. The primitive features extracted from an image are the boundaries where sharp intensity changes occur. Local symmetry representations of shapes are defined on these boundaries. Smoothed Local Symmetries provide intuitive descriptions of elongated regions. However, they do not provide intuitively plausible representations for round regions. They are also degenerate and unstable on these regions. Therefore, a companion representation for round regions is required.

3.1. The input representation

A black-and-white camera image of a real world scene consists of grey-level values for all points in a bounded region of 2-space. In human visual processing, images are interpreted as being composed of connected regions in which the grey level changes only gradually, together with a finite set of boundary curves across which the grey-level changes rapidly. Likewise, regions of 3-space and curves (e.g. boundary curves) seem to be represented in terms of regions of gradual change and boundaries where there is a sharp change. (Cf. Blake 1983a and 1983b, Asada and Brady 1984, Ponce and Brady 1985, Grimson and Pavlidis 1985.) Local symmetry representations, as well as many other shape representations, are defined on these boundaries of sharp intensity change. Note that representation of space to a finite degree of resolution in terms of connected sets and a finite set of boundaries is analogous to the representations of time and qualitative spaces used in Forbus (1984) and Allen (1984). The general idea used in both vision and practical reasoning work is that an infinitely dense reality can be represented by a set of regions where change is smooth or continuous and a set of interesting boundary points where change is abrupt.

It is possible that other features of a grey-scale image besides boundary locations might be used in shape description. The edge finder I use (Canny 1983) finds boundaries by detecting peaks in a first-derivative operator convolved with the image. This amounts to detecting zero-crossings of a second-derivative operator. Peaks in the output of a second-derivative operator have also been proposed as features for analyzing images. Watt and Morgan (to appear) argue that a representation for changes in intensity based on second-derivative peaks is more robust than one based on second-derivative zero-crossings. Mayhew and Frisby (1981) present evidence from stereo that second-derivative peaks as well as zero-crossings are used. Crowley (1982) also uses peaks, rather than zero-crossings to detect regions and edges in grey-scale images. In addition, color of regions seems to be useful in sorting out intuitive local symmetries from unintuitive ones, though the details of how this is to be done are not well-understood (see below, Chapter 5, Section 4).

The grey-scale input to the shape description system is digitized, so that a scene is described by a discrete array of pixels. The output of Canny's edge finder is an array of similar size in which each position is marked as to whether there is an edge there and, if so, what the orientation of the edge is. The most straightforward way to think about edge "points" is as small intervals, since they have not only a location but also an orientation. The boundary of a region is represented by a series of edge "points" that are connected in the 2-dimensional array. Each edge "point" (or interval) should be thought of as an interval overlapping the neighboring intervals in the curve. In this way, the set of edge "points" with orientations is a discrete representation for a connected curve.

3.2. SLS definition

The Smoothed Local Symmetry (SLS) representation for elongated shapes (Brady 1983, Brady and Asada 1984, Heide 1984, Connell 1985) is based on the notion of a local symmetry between two boundary points. Two boundary points A and B (see figure 1) are said to have a local (reflectional) symmetry[1] if the angle between the line AB and the outward normal at A is the same as the angle between AB and the outward normal at B. The symmetry center (C) for such a pair of points is the midpoint of AB. Another way to describe this situation is that the boundary around A and the boundary around B are locally reflectionally symmetric about the perpendicular bisector of AB.[2] In an elongated

[1] Brady et al. use the term "local symmetry". I will add the qualifier "reflectional" to distinguish this type of local symmetry from "local rotational symmetries," which I will define later in this chapter. I will use the term "local symmetries" to refer to both types of symmetries jointly.

[2] More precisely, in an infinitely dense description, the tangents are reflections of one another and thus the boundaries approach being reflections of one another as one considers smaller and smaller neighborhoods of the points A and B. If boundaries are represented with a discrete set of intervals, i.e. locations with orientations, then two intervals A and B will have a local

region, a point on one side of the region and the point which is perceived as being directly opposite it on the other side will have a local reflectional symmetry and the midpoint of the line joining the two points will lie on the perceived axis of the region. Current implementations of Smoothed Local Symmetries also place constraints on the color of the regions to both sides of the boundaries involved in a local symmetry. Since the form of the constraint used in current implementations seems to be perceptually incorrect and it is unclear what a more correct statement of the constraint would be, I defer discussion of it to Chapter 5.

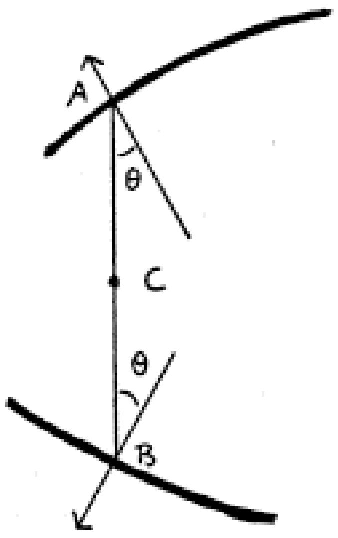

Figure 3-1. SLS Geometry: A and B are said to have a local (reflectional) symmetry if the angle θ between the outward normal at A and the line AB is the same as the angle between the outward normal at B and AB. The midpoint of AB, C, is the symmetry center for A and B.

A Smoothed Local Symmetry region (SLS region) is formed by grouping local reflectional symmetry pairs and their centers into connected curves.[3] Thus, there are two curves of boundary points, each containing one point from each symmetry pair, that form the *sides* of the symmetry region. The centers of the symmetry

symmetry if their normals are sufficiently close to being reflections of one another.

[3] This version of the definitions is a reworked version of the definitions found in Brady (1983), Brady and Asada (1984), Heide (1984), Connell (1985). While the representations of elongated regions remain more or less the same as in these references, the details of the definitions that produce these representations are rather different. This description also abstracts away from details of the several implementations of algorithms for computing local symmetries.

pairs form the *axis*[4] of the symmetry region. This axis corresponds well with the perceived axis of the region. The line segments connecting corresponding pairs of symmetry points are called the *ribs* of the region and the length of a rib measures the local width of the region. The two-dimensional region covered by the symmetry region is the union of all of the ribs or, alternatively, the region bounded by the two sides and the first and last ribs. Figure 2 shows the boundaries, region covered, axis, and selected ribs for one symmetry region. The symmetries in a Smoothed Local Symmetry region must progress in a consistent direction along the axis. That is, the region cannot double back on itself, as illustrated in Figure 3.

The raw descriptions of a symmetry region computed from a digitized input image give the locations of the sides and axis and the width of the region for each of the discrete set of symmetry pairs that make up the symmetry region. This detailed information can be summarized into symbolic representations which describe various salient features of the region, including:

- the length and curvature of the axis;
- the average width and pattern of change in the width of the region;
- the ratio of the width of the region to its length, called the "aspect ratio";
- whether the region is part of the object being considered or part of the background (determined by the color of the the region in current implementations).

For more detailed description of descriptive parameters for SLS regions, see Connell (1985) and Heide (1984). The aspect ratio of a region can be used as a measure of how perceptually salient the region is. A region with a very high aspect ratio is typically not salient, whereas a region with a low aspect ratio is, as shown in Figure 4 (Also see discussion in section 6.3 "Locality of Computation".)

In analyzing a shape into symmetry regions, there are a number of features that indicate where there are boundaries between regions or subregions:

- an interior color boundary;
- a sharp change in one of the parameters of the region, e.g a sharp bend in the boundary or the axis, a sharp change in the width of the region, a sharp change in the derivative of the width of the region;
- a minimum of width. (This criterion has been proposed but is not used in current SLS implementations.)

[4] called the *spine* by Brady

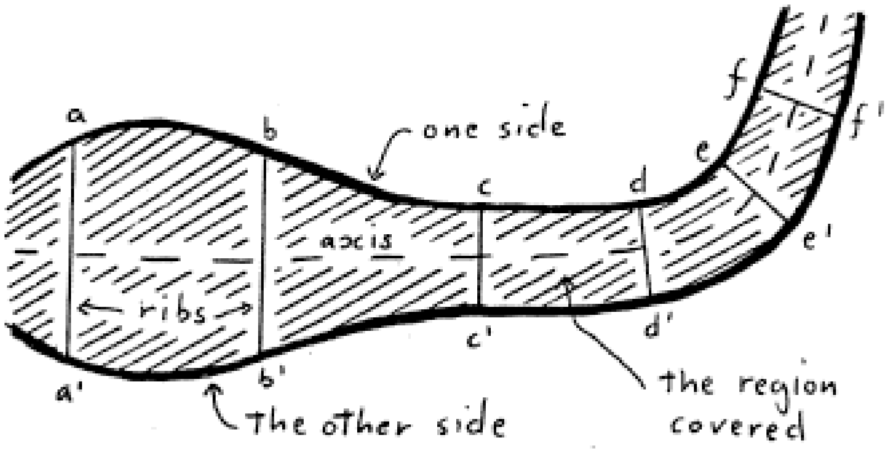

Figure 3-2. An SLS region with boundaries, axis, ribs, and region covered marked. The pairs of points connected by ribs, e.g. a and a', are corresponding pairs of points, i.e. pairs of points with local reflectional symmetries.

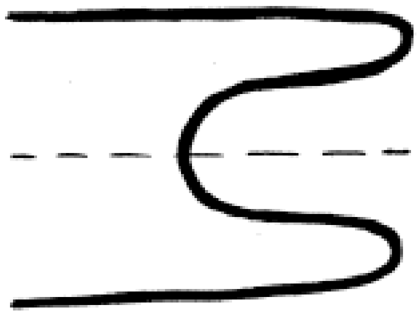

Figure 3-3. A hypothetical SLS region that doubles back on itself. This is not allowed. The symmetries in an SLS region must progress in a consistent direction along the axis. Thus, the boundary shown must be analyzed as two distinct SLS regions (a long region with an inlet cut out of its end).

Descriptions of a shape using symmetry regions should involve symmetry regions that are maximal given the above considerations, i.e. only break regions when there is some reason to do so. I do not know of any iron-clad rules for whether to break up a shape at a possible boundary. Obviously, this decision is partly determined by the strength of the boundary, e.g. the sharpness of a "sharp" change in a parameter. The decision also seems to involve functional and

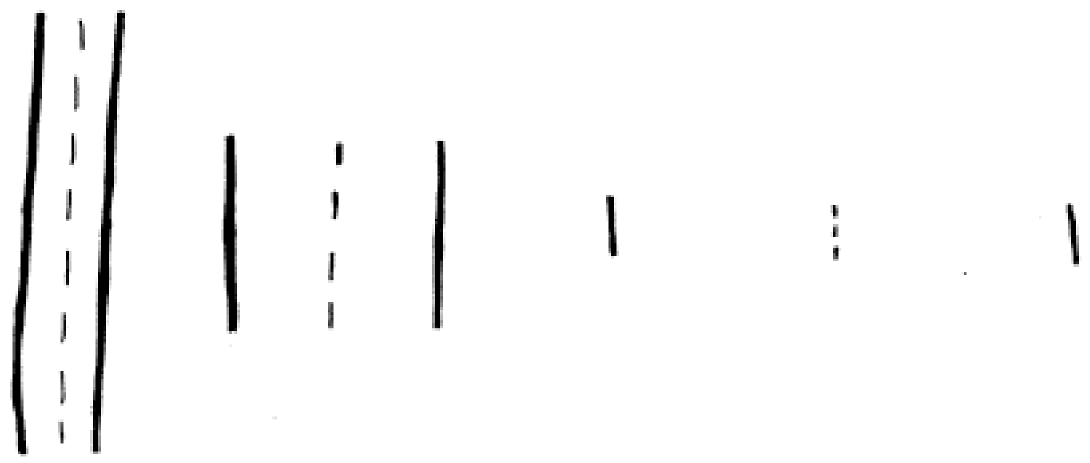

Figure 3-4. Three SLS regions with varying aspect ratios. The region on the left has a low aspect ratio and is perceptually salient. The region on the right has a very high aspect ratio and is not salient. The middle region is intermediate.

other high-level context. A given shape may admit of more than one plausible analysis in terms of subparts. For more discussion of these issues, see Connell (1985), Heide (1984), Brady and Asada (1984), and Hollerbach (1975).

In the process of describing an input shape using symmetry regions, the intuitively best representation may involve joining together two symmetry regions into one longer region, by creating sections of boundary that are not in the input image. There are two basic considerations involved in determining how good a joined region is:

- Fidelity to the data: the area of the join should be small compared to the areas of the regions being joined and the lengths of the added boundaries should be small compared to the lengths of the boundaries of the two original regions
- The combined region, including the added section, should form a good SLS region: the added points should form good local symmetry pairs and there should be no indication of a boundary in the join region, e.g. no sharp changes in parameters.

There may be more than one way to cover the same region of an input shape or scene using local symmetry regions. For example, a fat rectangle has an SLS axis along its main axis, and another one perpendicular to it. Both regions cover the entire area of the rectangle. A local symmetry description of a complex shape may also involve combining multiple local symmetry regions, either by joining them together or by cutting one out of another. This results in more more possibilities for multiple descriptions of the same shape. The basic consideration involved in building a local symmetry description is that:

- Except in the case of cutouts, a given 2-dimensional region with a given boundary is only described by one local symmetry region.

Actually, this statement of the criterion is somewhat vague and possibly not perceptually accurate. Revising it is a topic of current research and I will defer detailed discussion of it to Chapter 5 (Section 2).

Smoothed Local Symmetry representations can be computed robustly from input images and provide intuitive descriptions of elongated regions. For example, Figure 5 shows the Smoothed Local Symmetry regions of an airplane figure, computed by Brady and Asada's (1984) implementation of Smoothed Local Symmetries and Connell's (1985) analysis code.

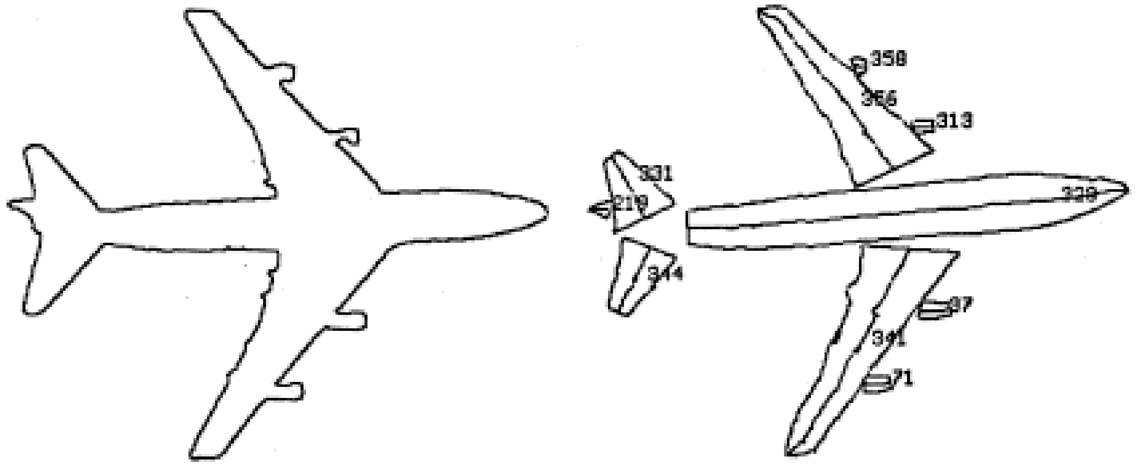

Figure 3-5. The boundary of an airplane (left) and the Smoothed Local Symmetry regions computed for it by the code described in Connell (1985).

3.3. Problems with the SLS

Although Smoothed Local Symmetry regions are good representations for elongated regions, they are not appropriate representations for regions which are round, nor for half-open regions bounded by one straight side, such as background regions extending to the edges of the image. Smoothed Local Symmetry representations of round regions are counter-intuitive, because such regions do not have a particular perceptually salient axis. Half-open regions do not in general have a Smoothed Local Symmetry representation at all, because they only have a boundary on one side. Further, in both of these cases, the Smoothed Local Symmetry representation has an infinite degeneracy and is unstable under small changes to the shape.

The Smoothed Local Symmetry representation is infinitely degenerate and unstable on round regions. First, all pairs of points in a circle or partial circle have local symmetries between them. Thus, a circle has infinitely many distinct SLS representations, each in terms of a different axis. It is necessary to explicitly detect circular regions in order to avoid computing the large numbers of possible Smoothed Local Symmetry regions within them. (Current implementations use various heuristics to avoid computing more than a few symmetries within round regions.) An axis-based description, such as Smoothed Local Symmetries, does not provide perceptually appropriate representations for round regions. A round or roundish shape does not have one perceptually salient axis. Rather, such a shape is most naturally described in terms of a center and angle/radius location of points on the boundary, relative to the center. Unless there is influence from outside context, the possible local symmetry axes of a round region are not in general perceptually salient (for details, see below, section 3.9).

The SLS representation is also unstable on regions which are close to circular. A region which is close to circular, but not quite circular, will in general have only a small number of possible symmetry axes, but extremely small changes to the shape of the region will drastically change which axes are possible. Furthermore, in going from a circular region to one which is perceptually very close to circular, there is a sudden change from a representation with infinitely many axes to one with only one or two axes, as illustrated in Figure 6. This instability will occur when noise or texture is added to a circle, as well as when a circle is deformed, e.g. into an oval.

Smoothed Local Symmetries also do not provide any representation for half-open regions which have a single boundary that is straight or perhaps has a shallow curve. Such regions occur when a background region or an object extends beyond the edges of the current field of view, as shown in Figure 7. David Braunegg has also pointed out to me that in representing 3-dimensional shapes

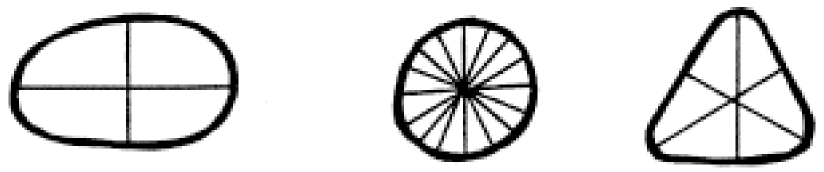

Figure 3-6. A circle with selected axes and two deformations of a circle, with all axes. If you transform the oval smoothly into a circle and then smoothly into the triangular figure, there will be a sharp change in representation as you pass through the circle.

we need some representation for regions for which we can only see one of the bounding surfaces. Although the full model for a 3-dimensional flat object (e.g. a table top) will have two bounding surfaces and the 3-D equivalent of a Smoothed Local Symmetry region, partial models built up from one view of an object may only have evidence for the shape of one of the bounding surfaces.

In addition to this lack of representation for half-open regions bounded by straight lines, the Smoothed Local Symmetry representation is infinitely degenerate and unstable on straight lines in much the same way that it is degenerate and unstable on circles. In an exactly straight line, every pair of points has a local symmetry, with the boundary segments exactly perpendicular to the axis of the symmetry. These symmetries are not perceptually salient. Further, the SLS representation is unstable on boundaries that are close to straight lines: small deviations from a straight line, due to noise or bending, will result in drastic changes to the possible symmetries, as illustrated in Figure 8. Again, symmetries between points in a boundary that is close to straight are not in general salient and current implementations use various heuristics to avoid computing such symmetries.

Thus, in addition to the Smoothed Local Symmetry representation for elongated regions, we need representations for round regions and for lines that are close to straight (bounding half-planar regions). In this thesis, I will describe Local Rotational Symmetries, a representation for round regions. (A representation for boundaries that are close to straight will have to wait for future research.)

3.4. LRS definition

The Local Rotational Symmetry representation for round regions is defined in a way parallel to the preceding definition of Smoothed Local Symmetries, with a few modifications required by differences in the two types of symmetries. I first define a rotational version of a local symmetry. A Local Rotational Symmetry

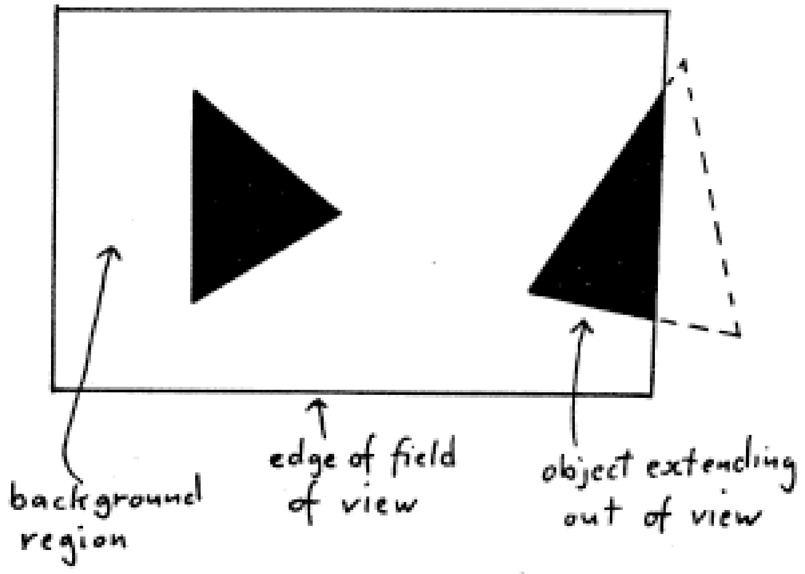

Figure 3-7. Examples of half-open regions: a half-open background region bounded by one straight side and an object extending out of the field of view.

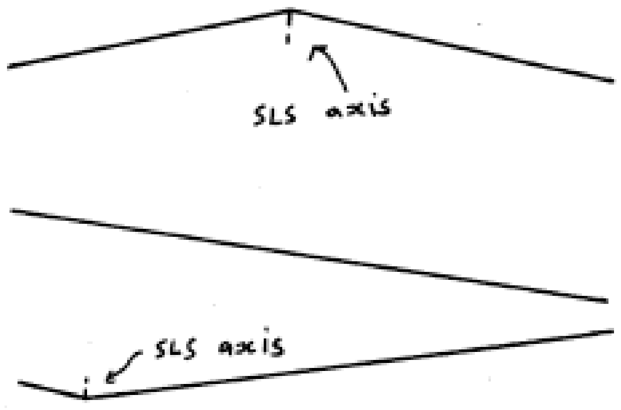

Figure 3-8. A straight line and two slight deformations of the line. The bent lines (top and bottom) each have one SLS axis. The straight line in the middle has either infinitely many SLS axes or none. As one bent line is smoothly transformed into the other via the straight line, the possible local symmetries of the figure change drastically. Thus, the SLS representation is unstable on straight lines.

region can then be defined as a region bounded by a connected set of boundary points with local symmetries around the same center.

A Smoothed Local Symmetry region is a region whose boundaries are connected curves that locally form reflectional symmetries about the axis. I will define a *Local Rotational Symmetry* region to be a region whose boundary is a connected curve that locally forms a rotational symmetry about the center of the region. In other words, if a section of the boundary is rotated about the center, it "comes close to matching" the next section of boundary. More precisely, I will say that a boundary point has a *Local Rotational Symmetry* (LRS) to a center location if the normal to the boundary at that point at a small angular distance from the radius from the boundary point to the center location, as shown in Figure 9. This condition on the normals at boundary points guarantees that if you map a section of the boundary onto another nearby section of boundary, each boundary point should be displaced in a direction approximately tangent to a circle around the center location, i.e. the mapping was approximately a rotation around the center location.

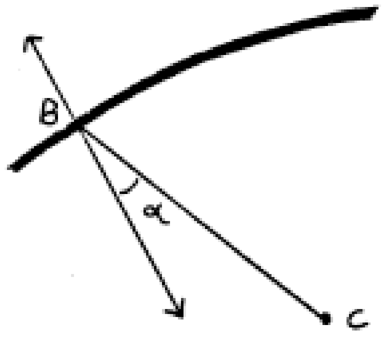

Figure 3-9. A boundary point B is said to have a local rotational symmetry about a center point C if the angle α between the radius BC and the normal to the boundary at B is small.

A Local Rotational Symmetry region (LRS region) is a region bounded by a connected curve of boundary points, each of which has a Local Rotational Symmetry with a common center location. So, Figure 10 shows a typical LRS region. I will refer to a line segment connecting a boundary point to the center as a *radius*. Figure 11 shows some LRS regions. Obviously, a circle is an LRS region. The other regions illustrate various types of minor deformations of a circle, all of which are perceptually more or less "round" or "not elongated". I will allow the boundary of an LRS region to be an open curve and I will also allow the boundary to spiral. For example, the bump, dent, spiral, and round end shown in Figure

12 are all LRS regions. Open boundaries whose ends can plausibly be joined are perceived as closed boundaries with gaps. However, when the ends are too far apart, the ends are not joined, and the boundary is perceived as bounding only a partial round region. However, the boundary should not "back up", i.e. switch direction of rotation, as shown in Figure 13.

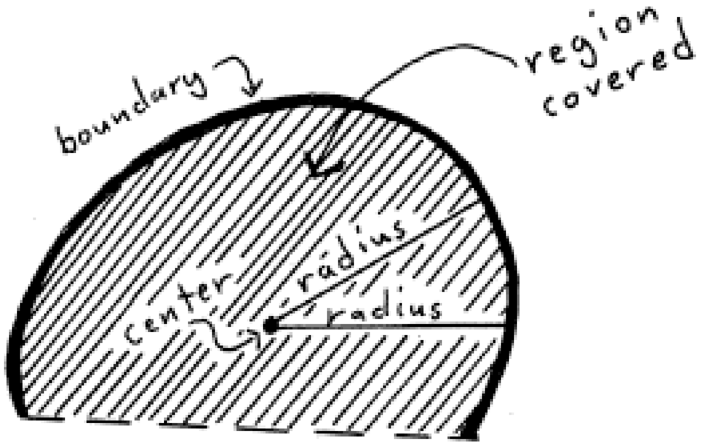

Figure 3-10. A typical LRS region, showing the boundary and center of the region, the 2-dimensional region covered, and selected radii from the center to the boundary.

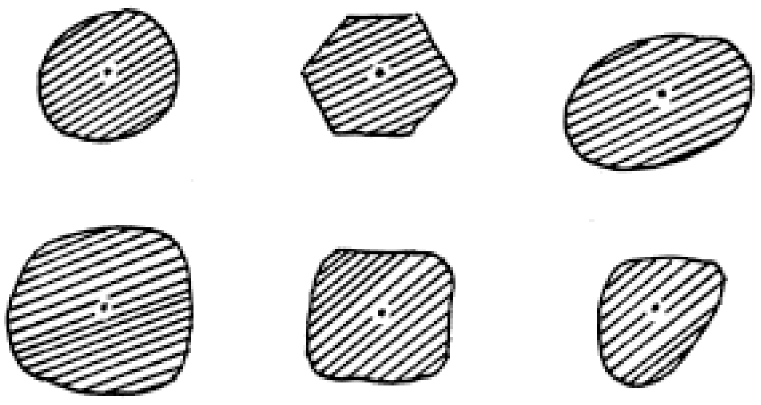

Figure 3-11. Examples of LRS regions, with the centers of the regions marked.

The two-dimensional region covered by an LRS region whose boundary is closed is obviously the region inside the boundary curve. If the boundary curve is

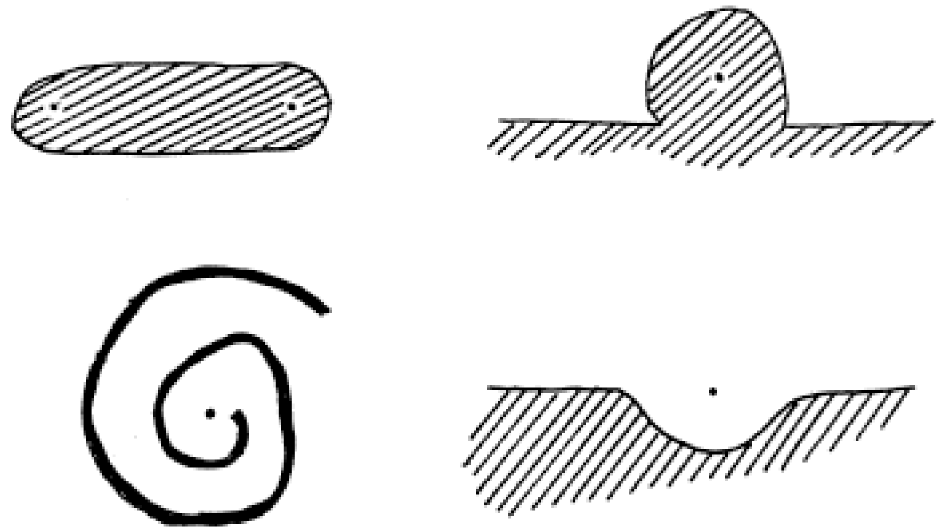

Figure 3-12. Examples of open LRS regions: a bump, a dent, a spiral, and the round ends of a rod.

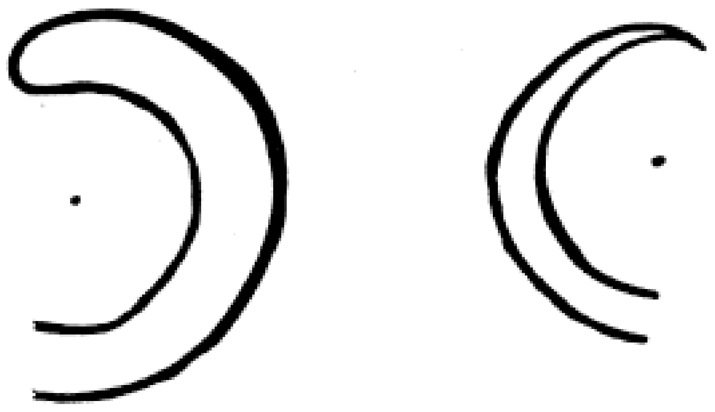

Figure 3-13. An LRS region is not allowed to switch direction of rotation. Thus, each of the two examples shown must be represented by two distinct LRS regions.

open and does not spiral, the region covered is the area bounded by the boundary curve and a straight line joining the two ends. In other words, the region covered is the union of all line segments joining two of the boundary points, as shown in Figure 10. The region covered is not the union of the radii of the LRS or the area bounded by the boundary curve with the ends joined by a smooth curve, e.g. a smooth interpolation of the radii. As shown in Figure 14, the regions specified by these definitions do not match human perceptions. A spiral LRS does not bound a region in a coherent way. For some purposes, the area of a spiral region may be defined by closing the open end of the spiral and taking the whole interior. I am not sure whether this is the only perceptually reasonable definition of the region bounded by a spiral.

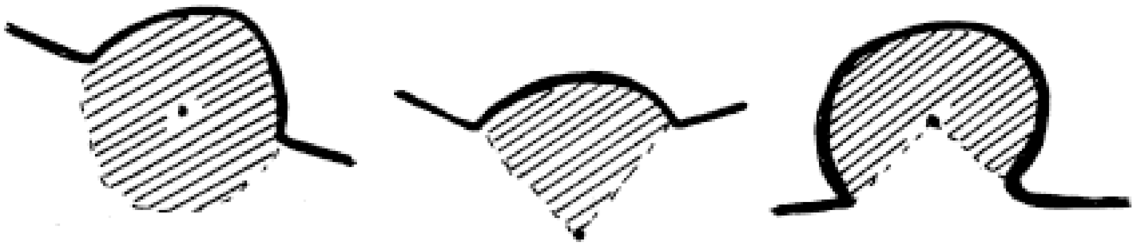

Figure 3-14. The left-hand example shows the region bounded by a partial round boundary plus a smooth join of its ends. The middle and right-hand figures show the regions obtained using the union of the radii of a partial round region. Neither of these definitions matches the perceived area of a partial round region.

In a Smoothed Local Symmetry region it is possible to require that the normals at each pair of corresponding points be exactly reflections of one another in the theoretical dense model. In real data, some allowance for noise and digitization error is required, perhaps 5-10 degrees. However, in a round region, it is not possible to require the normals to be exactly the same as the radii to the center, even allowing for noise and digitization. Most of the points on the boundary of a region that seems intuitively "round" have normals closely aligned with the radii, but a few points can be relatively far from perfect symmetries, as shown in in Figure 15. In both types of symmetries, good regions have both long boundaries and symmetries that are close to exact. In a Smoothed Local Symmetry region, since a symmetry center is only shared by two points, these two desirata are relatively independent. In a Local Rotational Symmetry region, since the center is shared by a large number of points, an optimal solution involves a compromise between increased length and increased divergence.

Since closed LRS regions can contain points with non-exact local symmetries and since the boundaries of round regions can be open, it requires no extra machinery to allow the LRS representation to handle spirals. In fact, it would require extra machinery in the theory to forbid them. The difference between a spiral

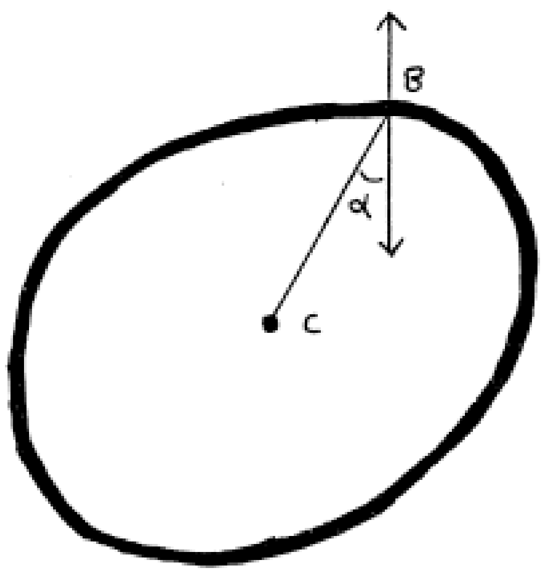

Figure 3-15. Individual points on a round region can have normals that are relatively far from being aligned with the radius to the center. For example, the angle α between the normal at B and the radius BC is about 40 degrees.

and a round region of more or less constant radius is analogous to the difference between pointed SLS regions and more elongated ones. Both spirals and round regions are intuitively described in terms of a center and radii from it to a connected boundary. Variation in radius *per se* does not distinguish spirals from round regions. The variation in radius of a perceptually round ellipse can be as much as the variation in radius of a spiral. In fact, sections of a spiral whose ends do not come close to the same angle form reasonable half-round regions. For example, the boundary of the bump shown in Figure 12 is actually part of the spiral shown in the same figure. A spiral could be analyzed as a set of partial round regions, each one extending until its ends get close enough together that they can be seen not to meet. However, on a regular spiral, there are infinitely many places to break up the spiral into partial round regions and there is no principled way to choose among them.

Spirals can occur in natural figures, e.g. a spiralled tail or a coil of rope or vine tendril, and in line drawings. Since they do not seem to be difficult for people to understand, spirals should have a relatively simple shape representation. Similarly, a spiral is a common way of mis-drawing a circle or an "O", as illustrated in Figure 16. While the defect should be noticed, it seems a perceptually small difference between basically similar figures. Spirals seem to be perceptually similar to round regions, except that they overlap (or come close to overlapping) the same angle with different radii. If spirals were not allowed to be round regions, then we would have to add a special shape representation for them. Fortunately, the best representation for round regions automatically provides simple descriptions of spirals. Furthermore, the representations of round regions in terms of radius/angle locations of boundary points provide the information needed to identify spirals as special for later processing, by detecting that they cross the same angle twice.

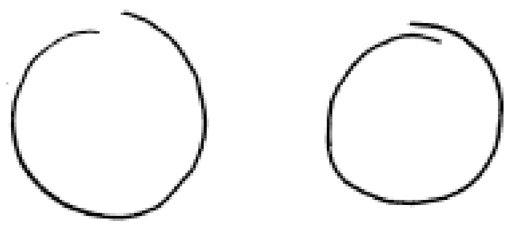

Figure 3-16. Two spiral mis-drawings of a circle or an "O". While the difference should be noticed, both figures seem perceptually close to a circle and should have a representation similar to that of a circle.

3.5. The optimization problem

Since Local Rotational Symmetries are not exact, a given round region has, in general, not only a plausible LRS analysis using the perceived center of the region, but also plausible LRS analyses using centers near the perceived center. These sub-optimal LRS analyses are not salient. Thus, picking out just the perceived center of a round region requires finding *locally optimal* pairs of a center and a boundary. There can be more than one SLS analysis of a given region. For example, Figure 17 shows two different SLS analyses of an ellipse. However, multiple SLS analyses are generally a small set of qualitatively different analyses. This is because Smoothed Local Symmetries are already an exact symmetry and thus typically locally optimal.[5] Choosing among a discrete set of qualitatively different alternatives is somewhat different from picking locally optimal choices from a set of alternatives that vary smoothly.

I use several criteria in deciding how good an LRS region is:

[5] But see the discussion of SLS's within region regions in Section 3.9 (below).

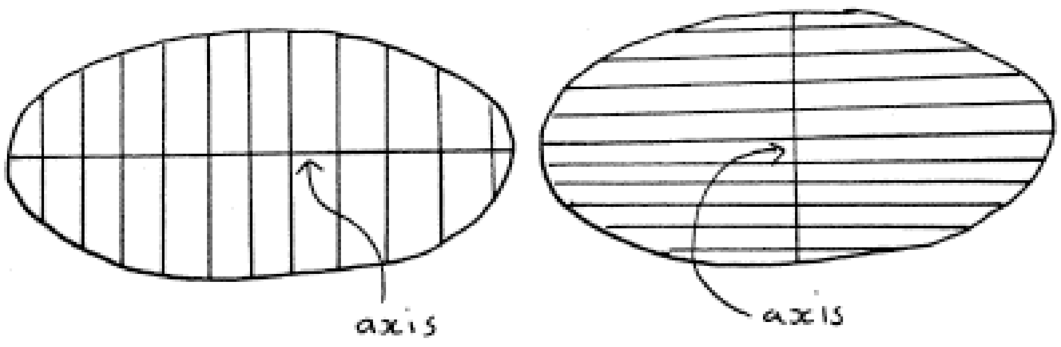

Figure 3-17. Two SLS analyses of an oval in terms of different symmetry axes.

- The average angular deviation between the normal at each boundary point and the radius from the center to that boundary point should be low;
- The region should have a long angular length;
- Closed boundaries are prefered to open boundaries and spirals are disfavored compared to either.

For example, Figure 18 shows a set of ovals with increasingly bad average angular deviation around their centers (assuming one center covering the whole boundary in each). I use the average deviation rather than a threshold on individual deviations, because the presence of boundary points with small deviations seems to license some boundary points with large deviations, as in the example shown in Figure 3-15. Figure 19 shows examples of open boundaries with varying angular length. Note that angular length should be used rather than absolute length of the boundary, because angular length is invariant under changes in the size of the region. A low average deviation of normals within a region seems to be able to partially compensate for a small angular length. That is, partial but highly regular regions are as salient as more complete but irregular regions. Chapter 4 discusses details of exactly how regions are evaluated.

3.6. Joins and segmentation

Like the boundary of an elongated region, the boundary of a round region may be broken up into disconnected sections by attached or occluding parts, or by sections cut out of the region. For example, Figure 20 shows the boundary

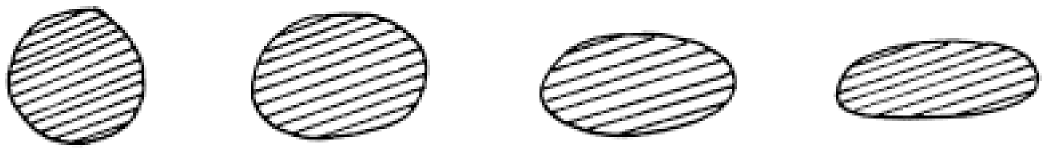

Figure 3-18. A set of ovals that are increasingly flattened. As the ovals are flattened, the average angular deviation increases and the region becomes less good as a round region.

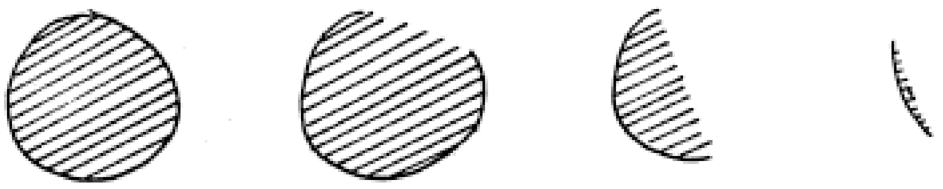

Figure 3-19. A set of boundaries with increasingly small angular length. As the angular length gets smaller, it seems less and less likely as the boundary of a round region.

of a spanner wrench. Each end of the wrench is perceived as one round region, though the boundary of the end is broken up into two sections by the handle and by the square cutout. In describing the round regions in such a figure, one must join together disconnected sections of boundary to form one connected boundary. In addition, the ends of a connected open boundary should be joined if possible. For example, the two pieces of boundary of each round end of the spanner wrench should be joined across *both* gaps to make a closed boundary.

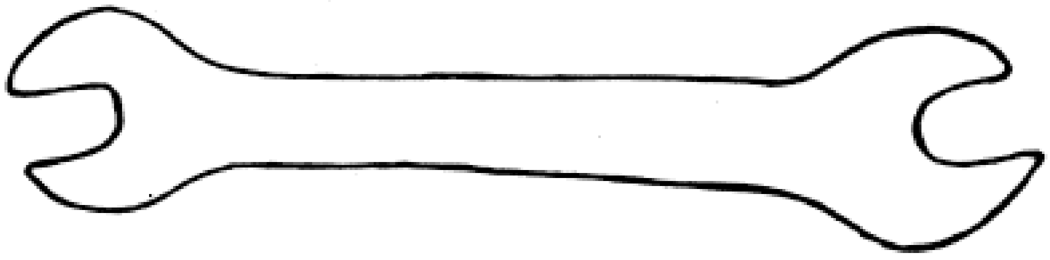

Figure 3-20. The boundary of a spanner wrench. The boundaries of the round ends of the wrench are broken up by the attached handle and by the square cut-outs between the jaws.

When the boundary of an LRS region is made up of two disconnected sections of boundary joined together, the resulting LRS region, including the boundary points hypothesized in the join, must meet all the normal conditions for an LRS region. In particular, the angular deviation between the normals along the new section of boundary and their corresponding radii to the center must be low and the direction of rotation must stay the same for the whole boundary. Also, there is an additional consideration in evaluating boundaries with joins: fidelity to the input data. Whether the combined boundary formed by joining two boundaries is better than the two separate boundaries is a tradeoff between the increased length and the degradation in goodness caused by adding hypothetical points, particularly if these points also have high deviations. The situation is similar when the ends of an open curve are joined to form a closed curve, except that there is the additional factor that closed boundaries are more highly favored, all other things being equal. Figure 21 shows a series of boundaries that can be closed with joins of similar length, but varying average deviation from normal.

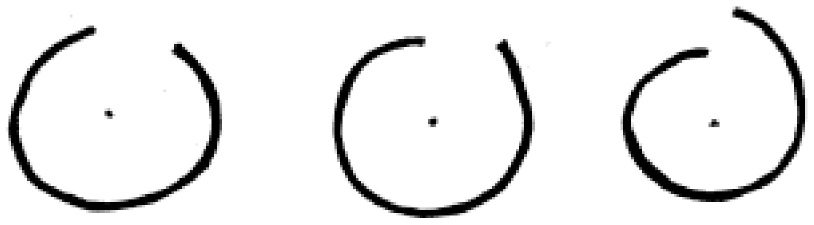

Figure 3-21. A series of boundaries to be closed. The points required to join the leftmost boundary would all have small deviations and the join seems good. The points required to join the rightmost boundary would have abysmal deviations and the join seems bad. The middle boundary is intermediate.

One definition of fidelity to data would be to use the percentage of real points in the joined boundary. Figure 22 shows a series of boundaries with varying percentages of real points. Another consideration is a more local measure of fidelity to the data: how long the gap is compared to the lengths of the two boundaries. The difference between these two measures occurs when the boundaries to be joined are of very different lengths. Even though the longer boundary may be long enough to guarantee a high percentage of good points in a combined boundary, it seems that the size of the gap must also be short compared to the length of the smaller boundary in order for the join to be perceptually good, as illustrated in Figure 23. The current implementation uses a combination of both criteria. It now seems to me that local fidelity by itself may be sufficient to account for the data. Finally, there seems to be a constraint that two boundary ends cannot be joined if the gap between them is more than about 40-50 degrees: a boundary with a gap larger than that seems to be perceived as a partial round region, rather than a full round region with a gapped boundary. Figure 24 shows a series of open boundaries with varying length gaps between the ends, illustrating that open boundaries with long gaps are not closed. Pinning down the details of these constraints is a matter for further research.

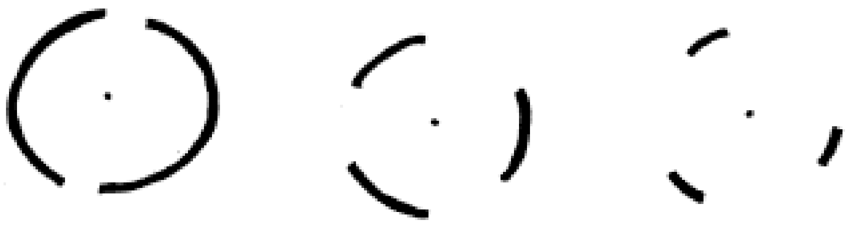

Figure 3-22. A series of boundaries with varying percentages of real points compared to points that would need to be hypothesized to join them into a complete boundary. The set of boundaries on the left, with high percentage of real points, can plausibly be joined. The set of boundaries on the right, with a low percentage, cannot. The middle set is intermediate.

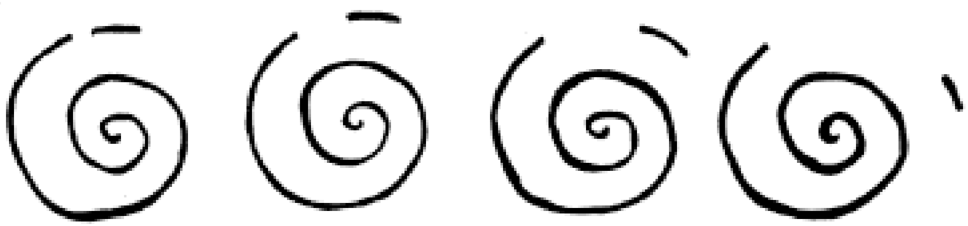

Figure 3-23. A series of boundaries with varying length gaps compared to the length of the shorter boundary. The join gets worse as the relative size of the gap increases. The join on the rightmost figure does not seem plausible because it is too long relative to the length of the shorter boundary, although the overall percentage of good points is high.

On the other side of the coin, a sudden change in parameters of a round region, such as a corner, is a reason for breaking up the region into sub-regions. When a Smoothed Local Symmetry region is broken up into sub-regions, each sub-region has the same status as any independent region. However, when a Local Rotational

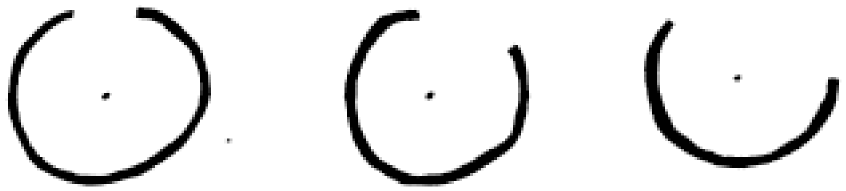

Figure 3-24. A series of open boundaries with varying length gaps between the ends. The leftmost boundary can plausibly be closed. The rightmost boundary cannot plausibly be closed. The middle boundary is intermediate.

Symmetry region is broken up into sub-regions, the subregions may retain the global center of the region, even when that is not the optimal center for the sub-region taken in isolation. For example, consider a hexagon. The corners are salient and seem to divide the boundary of the region into subsections. However, this division does not affect the perceptual center of the hexagon, which is the center of the entire boundary, not the perceptual center of any of the sides in isolation.

3.7. Results and symbolic descriptions

I have implemented an algorithm for computing locally optimal LRS regions. The details of this algorithm are described in Chapter 4 and results of this algorithm on sample figures are shown in Section 4.7. For each region, the program computes a center location and a full ordered list of boundary point locations relative to the center. Radii from the center to boundary points and orientations of boundary points relative to the center can be computed extremely quickly.

From this output, one could compute a symbolic representation of the region. Such a description should include facts such as the following:

- Location of the center of the region;
- Range of angles covered as one moves along the boundary (If the boundary spirals, this includes noting which angles are duplicated and how many times. The boundary is *not* parameterized by angle.);
- Which sections of the boundary are real boundary points and which were parts of joins;

- Whether the boundary is open or closed;
- Whether the boundary spirals;
- Average, minimum, and maximum deviation of normals at boundary points from corresponding radii to the center;
- Average, minimum, and maximum radius and locations of extrema in the radius;
- Description of simple patterns of change in radius, e.g. generally increasing, approximately constant, ratio of minimum to maximum radius;
- Locations of sharp changes in the radius.

These parameters for symbolic description of LRS regions are similar to the parameters proposed for Smoothed Local Symmetry regions in Brady (1983), Brady and Asada (1984), Heide (1984), Connell (1985). The current implementation of Local Rotational Symmetries does not yet compute these descriptions, although it computes some of these parameters (e.g. angle range covered) in the course of evaluating regions.

3.8. Alternatives

My definition of Local Rotational Symmetries specifies that the crucial properties that determine when a set of curves bounds a round region about a particular center are the *connectedness* of the boundary and the *angular deviation* between the normal at each boundary point and the radius from the boundary point to the center of the region. I have formulated the problem of finding perceived regions as an optimization problem, in which optimal regions (pairs of a boundary and a center) are determined by a compromise between low angular deviation and long angular length of the boundary. I considered a variety of alternative ways of defining round regions in the course of choosing this definition. The approach I used was to start with the properties of a circle, the round region *par excellence*, and observe which of these properties were preserved in regions which people perceive as slight deformations of a circle. In particular, methods of finding region centers using exact constructions such as intersecting normals, prove to be too sensitive to deformation.

Some familiar properties of a circle are:

- the curvature of the boundary is constant;
- all points on the boundary are equidistant from the center;
- the normal to the boundary at every point passes through the center;
- the boundary is a connected, closed curve.

The only one of these properties that holds exactly of round regions which are not circles is the connectedness of the boundary. When pieces of the boundary of a region are not connected in the input, it seems as if they are explicitly joined into connected boundaries during the process of shape description.

Exact normality is very sensitive to slight deformations in a figure. For example, in a fat ellipse or a hexagon, very few of the normals to the boundary actually pass through the perceived center of the ellipse. Intersections of normals are even less stable and can be relatively far from the perceived center of a nearly circular region. Furthermore, intersecting normals does not generalize well to 3-dimensional shape models, because non-parallel lines in 3-space are not guaranteed to intersect. Therefore, although normality to the center is a good indicator of the plausibility of a piece of boundary as part of a round region, *exact* normality must be relaxed. Since the shape representations should be invariant under changes in size, the appropriate measure of deviation from normal is angular deviation.

Local curvature is also very sensitive to slight deformations. For example, locally flattening a circle, e.g. to form a hexagon causes the local curvature to vary substantially from the perceived curvature of the region as a whole. Local curvature could be used as an additional, rough constraint on region curvature. However, local curvature along the boundary of a region does not seem to be the primary influence on where the perceived center of the region is.

Another possible definition of a round region would be to use some form of equidistance of boundary points from the center. Exact equidistance is sensitive to noise and deformation in much the same way that normality is, so any use of equidistance must involve approximate equidistance. Equidistance, by itself, is a poor indicator of how round a region is. At the very least, it has to be coupled with some connectedness constraint on the boundary, in order to keep concentric partial circles from scoring as well as an extremely circular ellipse. In fact, since even blatantly non-circular ellipses are perceived as relatively round, it will not work to impose a requirement of global equidistance, i.e. that all points in the region must be approximately equidistant from the center.

The contraint that seems to best reflect human perceptions is a requirement that points be *locally* close to equidistant from the center. Since the representation should be size-invariant, the measure of equidistance should be normalized for the size of the region. Given that the boundaries of regions form connected curves, requiring that the local change in radius normalized by the radius be small is equivalent to requiring the boundary to be approximately normal to the radius to the center.

It is not possible to salvage any of the exact solutions by smoothing the image or the boundaries first, at least not by using any smoothing technique that I am aware of. The problem is that smoothing is only effective at removing detail that is much higher-frequency than the feature being detected. Thus, smoothing can remove the effects of edge texture, such as serration, and image noise. However, smoothing to remove deformation at about the same resolution as the features you want to detect destroys the very structure you are looking for. For example, the round end of a pear, as in Figure 25, is typically not exactly circular, but smoothing it enough to make it circular will also smooth it into the pointed end of the pear. In general, if a round region is attached to other regions of similar scale, smoothing its boundary enough to regularize deformations in the round region will also smooth its edges into the adjacent regions. Techniques such as the ones described in Ponce and Brady (1985) and Grimson and Pavlidis (1985) to avoid bleeding of smoothing into adjacent regions cannot be used unless the region boundary can be detected before smoothing. The boundaries of round regions often connect smoothly to the boundaries of adjacent regions, as in a pear shape, so there is no obvious way to detect region boundaries prior to describing the regions. Furthermore, if the boundary of the region is broken up into disconnected sections by attachments, smoothing these sections of boundary individually will not correct for any discrepancies between local curvature in each fragment and the overall curvature of the whole region. A good example of this is the outline of a lemon shown in Figure 25.

3.9. Fixing the SLS infinite degeneracy

One of the reasons for developing a new representation for round regions was because Smoothed Local Symmetry representations have infinite degeneracies in

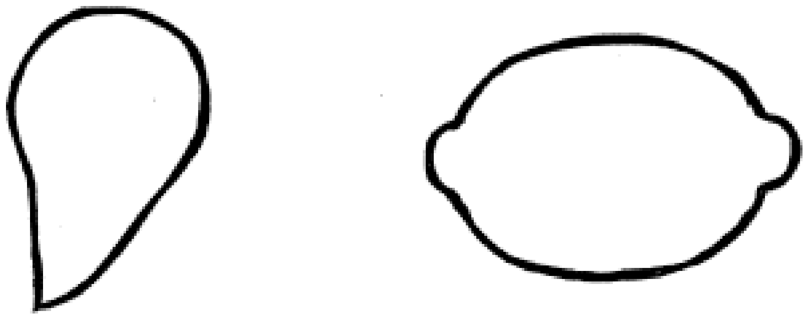

Figure 3-25. Outlines of a lemon and a pear. Smoothing the pear to eliminate deformations in the round end will smooth the round region into the pointed end. It is not obvious how to eliminate this bleeding of smoothing across the region boundary, since there is no obvious marker of the boundary location. In the lemon image, the region boundaries do occur at points of high curvature, which can easily be detected. However, if the two halves of the boundary of the body of the lemon are smoothed individually, this will not correct for the overall flattening of the region.

round regions. In addition to having a representation for these regions, we also need to suppress Smoothed Local Symmetries within these problem regions. A round region such as a circle has an infinite number (or a very large number in a finite-resolution representation) of possible SLS representations. Rather than computing all of these possiblities explicitly, we should detect these cases and *summarize* the possible analyses. Such a summarization would allow one to make one of the possible SLS analyses salient when external context provides some means of selecting among them. For example, in the shape in Figure 26, one axis of the round region is made salient by the fact that it is in line with the axes of the long sections attached to the round region.

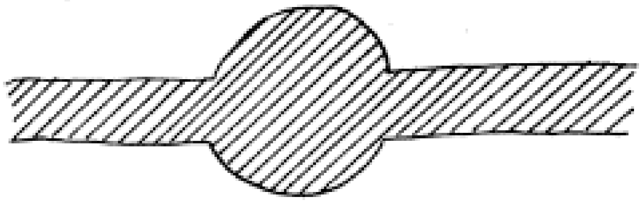

Figure 3-26. A round region attached to two elongated regions. The axis of the circular region that is in line with the elongated regions is made salient by their presence.

Forbidding all Smoothed Local Symmetries within an LRS region does not produce the perceptually correct type of suppression. For example, when there is a corner in an LRS region, the pointy SLS describing the corner is salient. Similarly, when an LRS region has two parallel flat sides, the SLS describing their relationship may also be salient. Figure 27 shows examples of this. Similarly, there is a salient SLS between adjacent sides of the spiral in Figure 28, although they form part of the same round region.

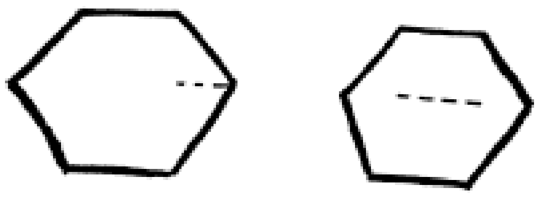

Figure 3-27. A hexagon. The Smoothed Local Symmetries in the corners (left) and between opposite sides (right) are salient, despite the fact that a hexagon is an LRS region.

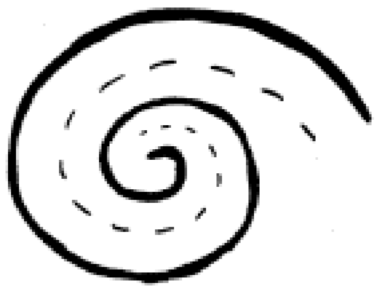

Figure 3-28. A spiral. The Smoothed Local Symmetries between adjacent boundaries in the spiral are salient, despite the fact that the spiral is an LRS region.

What distinguishes salient from non-salient Smoothed Local Symmetries, in the absence of external context, seems to be that a Smoothed Local Symmetry is salient if it is *locally optimal*. That is, if you replace one of the boundary points of the symmetry with a point to either side of it on the boundary, the local reflectional symmetry between these two points will be markedly further from exact. The non-salient Smoothed Local Symmetries occur when a number of adjacent boundary points all have close to exact symmetries with the same opposite boundary point. Such situations can be detected during the computation of Smoothed Local Symmetries and these symmetries suppressed in a principled manner.

3.10. Future work and extensions

There are several obvious ways in which the symmetry representations presented above could be extended. First, I mentioned that Smoothed Local Symmetry representations have problems with straight line segments bounding half-open regions that are analogous to the problems with round regions. I have fixed the problems with round regions by developing a new representation for them. Perhaps the problems with straight lines and half-open regions could also be fixed by developing a representation for straight line segments and the half-planes they bound and by finding ways to detect the corresponding degeneracy during SLS computation.

Secondly, an analog of SLS and LRS representations should be developed for representing 3-dimensional regions of space. The obvious 3-dimensional analogs of the 2-dimensional symmetry regions are:

- round or sphere-like regions that are locally rotational symmetries about a point;
- elongated regions that are locally rotational symmetries about a line;
- flat regions that are locally reflectional symmetries about a surface;
- flat surfaces that divide 3-space (locally) into two open regions.

Finally, we need to develop a theory of how 2-dimensional local symmetries are computed on non-planar 3-dimensional surfaces. For example, the outsides of objects such as coffee mugs are often decorated with designs. These designs are essentially 2-dimensional figures, but the surface on which they are drawn is only locally planar. It may be possible to extend the definition of 2-dimensional local symmetries to allow them to be defined for any surface which is close to planar within the local area of the symmetry region.

Chapter 4: Computing local rotational symmetries

This chapter describes the details of an implemented algorithm for computing Local Rotational Symmetry regions and shows results of this algorithm on grey-scale images of objects.

4.1. Overview

The input to the algorithm is a grey-scale image. This image is smoothed and sampled to produce a series of finer and coarser resolution versions of the image. Boundaries of regions are extracted from each of these images by the edge finder described in Canny (1983). Computation of Local Rotational Symmetry regions proceeds from coarser to finer scales. At each scale of resolution, symmetries are computed exhaustively between all boundary points and all centers that are within a fixed search radius of one another. Thus, the exhaustive computation is restricted to being local. The reasons for imposing locality on the computation will be discussed in Section 6.3. Symmetries are also computed for centers and boundary points suggested by Local Rotational Symmetry regions found at the preceding (coarser) scale. The output of the program is a set of Local Rotational Symmetry regions at each scale of analysis.

Computation of Local Rotational Symmetries at one scale of resolution proceeds in three stages:

- For each center location (in parallel), compute a map of boundary points around the center, showing the angular deviation for each point;
- Compute the best regions for each center location;
- From among the regions computed for all center locations, choose the locally best regions.

Since the versions of the image at different scales are sampled at a rate proportional to the rate of smoothing, the entire computation is size invariant. That is, all parameters of the computation are constants that do not depend on the scale at which computation is being done. This computation is shown schematically in Figure 1.

The rest of this chapter will be a detailed discussion of how LRS regions are computed. The set of topics to be discussed include:

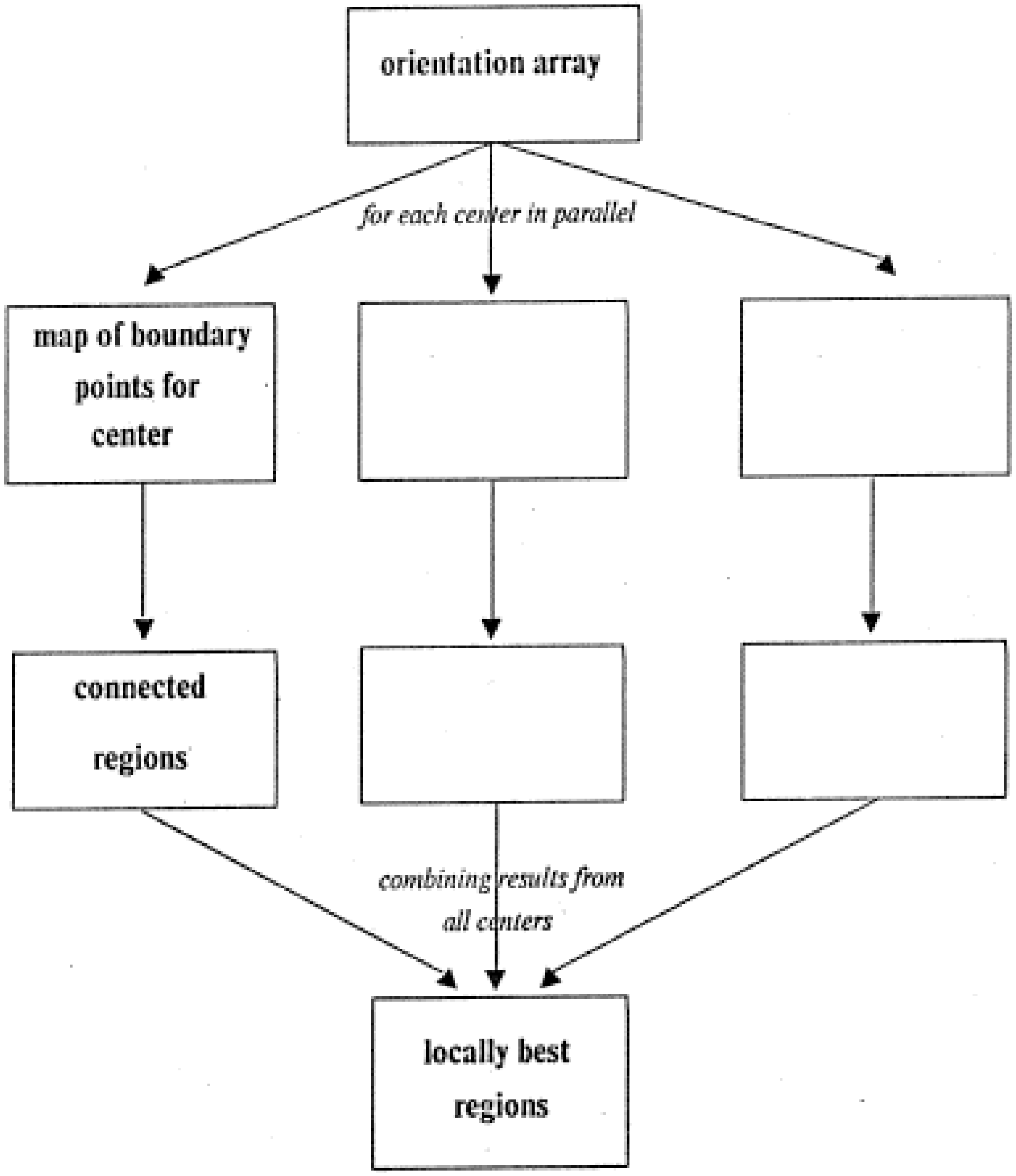
orientation array
for each center in parallel
map of boundary points for center
connected regions
combining results from all centers
locally best regions

Figure 4-1. Block diagram of symmetry computation for one scale of resolution. For each center location, the algorithm first computes a map of angular deviations for boundary points around that center. This map is then used to compute the best connected regions around that center. The results from all the centers are combined to find the regions that are locally optimal.

- the input
- computing the symmetry maps
- evaluating regions
- computing regions for each center
- non-maximum suppression
- evaluation of algorithm

4.2. Multi-scale input

I create representations of an image at different scales of resolution by smoothing the grey-scale image and sampling it at a rate proportional to the amount of smoothing done. (Chapter 6.1 compares this type of smoothing to other methods of creating representations at different resolutions.) Each image is half the area of the next-finer one and each pixel in the coarser scale version is a weighted average of the pixels around the corresponding location in the next finer scale version. The rate of sampling and the size and shape of the smoothing function used to transform the image at one scale into the smoothed image at the next scale are the same for all scale transitions. This is required in order that the representation be size-invariant.

The function used for smoothing the image is a Gaussian. In the current implementation, the Gaussian used has a σ of approximately $\sqrt{2}$ pixels and is approximated by an 11-pixel mask. The corresponding sample rate used for each dimension is $\sqrt{2}$. In other words, the area of the sampled image is half the area of the original image. The smoothing and sampling process starts with the original input image and continues until the dimensions of the smoothed image are smaller than the mask size. Images whose dimensions are unequal are padded to make them square as the smaller dimension approaches the mask size. This allows smoothing to continue until the larger dimension is smaller than the mask size. For detailed discussion of Gaussian convolution techniques, issues of noise, sample rates, and aliasing, see Crowley (1982) and Canny (1983). The set of smoothed and sampled images for a drain strainer is shown in figure 2.

From each of the smoothed grey-scale images in the set, I use the edge finder described in Canny (1983) to extract the locations of sharp changes in the intensity of the image. The output of the edge finder is a set of pixel locations at which there is an edge, together with the orientation of the edge at that point. In the

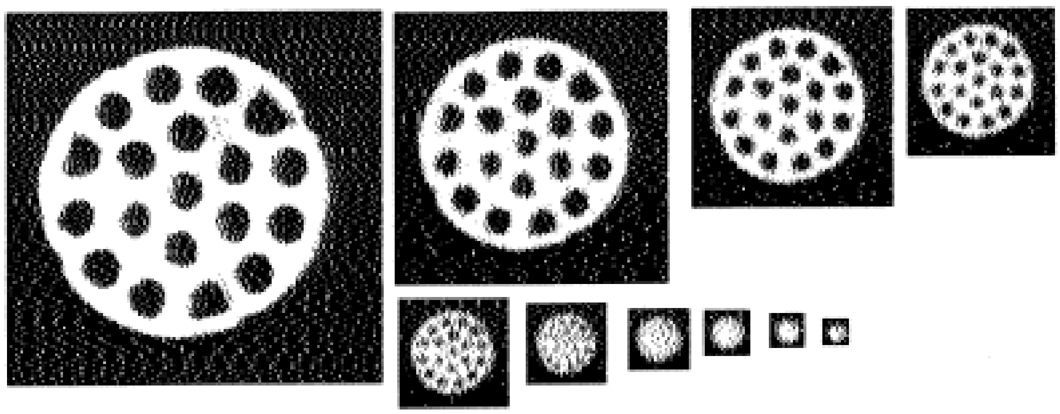

Figure 4-2. Smoothed and sampled images for a picture of a drain strainer. The largest grey-scale image is the original picture from the camera. Each of the other images was obtained from the previous one by smoothing with a gaussian and sampling.

current implementation, edge locations and orientations are stored in an array of the same size as the original image and the local symmetry calculations are done directly from this array. The edge finder returns locations where the edge strength is above a fixed threshold, with some hysteresis to encourage connected boundaries. It also provides information about the magnitude of the intensity change at each edge point, but I do not use this information. The edge finder described by Canny can be run with a range of mask sizes. Since running this edge finder with a larger mask size is equivalent to smoothing and sampling the image and running the edge finder with a smaller mask size, the smoothing and sampling done to create images at coarser scales achieves the same effect as changing the mask size of the edge finder. Thus, the edge finder is run with a single fixed mask size (8 pixels). The edges for the smoothed images in Figure 2 are shown in Figure 3.

The boundaries found by this edge finder are very high quality. Furthermore, the boundaries are thin. That is, exactly two of the eight neighbors of an edge location in the middle of a boundary will be marked as edge locations. Therefore, extremely simple algorithms can be used to extract connected boundaries from the set of edge locations. However, the boundaries from an image of an object are not guaranteed to be simple closed curves. There may be internal color boundaries in the image, producing 3-way joins in the edges. There may also be gaps in the boundaries, particularly at sharp corners of regions in the grey-scale image. In addition, boundaries may run off the edges of the image. These defects do not

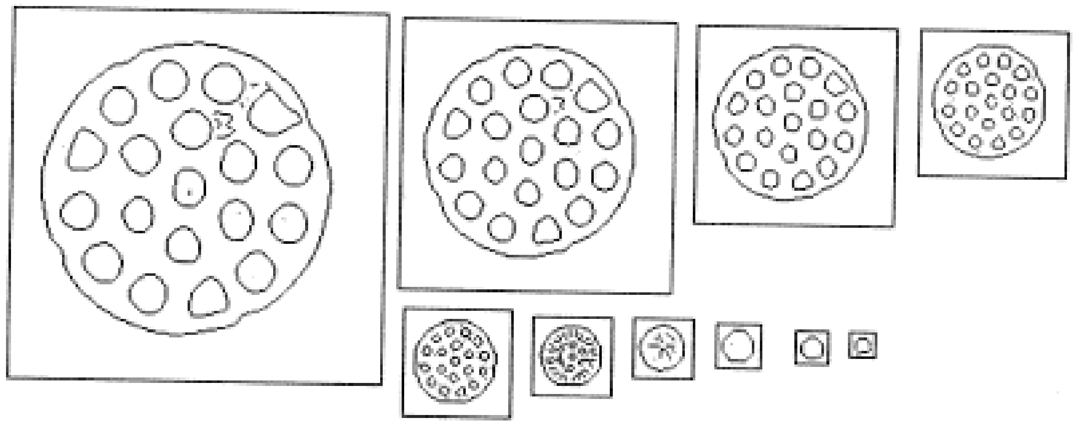

Figure 4-3. The edges extracted from the images in Figure 2 by the edge finder described in Canny (1983) These edges represent the locations of sharp changes in intensity in the grey-scale image.

necessarily represent problems with the edge finder: they occur in line drawings as well. We have not found any way to robustly resolve 3-way joins and gaps in a way that is consistent with people's judgements before the shapes of regions have been described. It may eventually be possible to resolve some of the gaps and joins in early processing, using facts about the orientation of boundaries that meet at an intersection or come near each other at a gap. However, it is also the case that representations of shape using smoothed local symmetries and local rotational symmetries can be computed without first resolving all gaps and joins and, indeed, the shape representations may provide additional information about how gaps and joins should be resolved.

4.3. Computing local rotational symmetries

Computation of local symmetries is done for all centers and boundary points within a fixed search radius of one another. Currently, the maximum distance between center and boundary point is 8 pixel units. In other words, in order to be detected, a round region must appear and get a good evaluation at a scale at which it is no larger than radius 8. If the region is a full round region with a closed boundary, this means that it will have approximately 50 points in its boundary. Increasing the exhaustive search radius allows the program to find increasingly more regions. These additional regions are regions which do not survive smoothing well, such as regions whose boundaries are disrupted by occlusions and thin regions such as rings and spirals. The output of the program is not sensitive to the exact setting of this parameter.

In addition, local symmetries are computed for pairs of centers and boundary points suggested by regions found at the next coarser scale. Given a location in a finer-scale image, the corresponding location in the next coarser scale image can easily be computed. Since edges may drift somewhat between scales, my program allows two extra pixels in each dimension for drift between two adjacent images (i.e. $2\sqrt{2}$ times the sampling rate in each dimension). The amount of drift allowed was determined by experimentation. Any edge in a corresponding location is considered a match for the coarse-scale edge, regardless of orientation. Since the LRS computation constrains the orientation of edges in LRS regions, this simplification is not a problem. In a more general setting, orientation would also have to be matched and more detailed figures for possible edge drift (possibly taking into account edge strengths) might be necessary (cf. Canny 1983). For each coarse-scale region, the sets of fine-scale locations corresponding to its center and the set of fine-scale boundary points corresponding to its boundaries are computed. Symmetries between these center locations and boundary points are computed at the fine scale.

Computation of local rotational symmetries about a given center location is done on a digitized map of boundary point locations around that center. For each boundary point, the algorithm computes the angular distance between the normal to the boundary at that point and the radius from the boundary to the center location, i.e. the amount by which this section of boundary deviates from an exact Local Rotational Symmetry. For example, consider a center location in the middle of an image of a lemon. Figure 4 shows the angular deviations for all boundary points in the figure, relative to this center location. Figure 5 shows the best connected boundary that the program computed for this center location.

Figure 6 shows the best connected regions for two other center locations in the lemon image. How well center locations account for a region decreases slowly as the location is moved away from the perceived center of the region. Center locations near the perceived center of a region generate candidate regions which have good evaluations, though not as good as regions generated by the perceived center. The righthand center location in the figure illustrates the fact that even locations somewhat displaced from the perceived center of a region will generate region descriptions similar to those generated by the perceived center. Center locations that are far from perceived centers do not generate regions with good

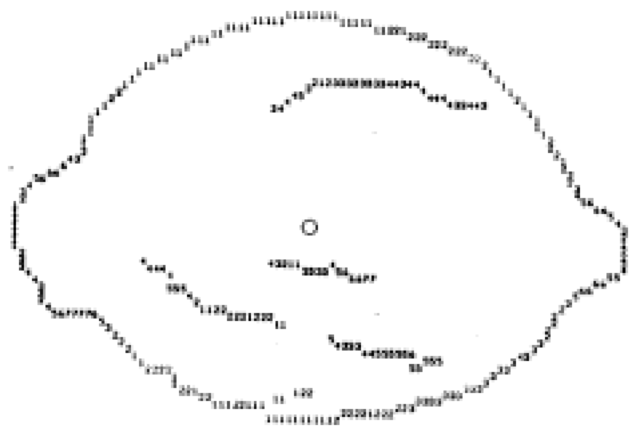
Figure 4-4. A map of angular divergences for the boundary points in a lemon figure, relative to the perceived center location of the figure (marked with a circle). At each boundary point, I have shown the angular divergence divided by 10. That is, a 2 in the figure indicates a boundary point with divergence between 10 and 20.

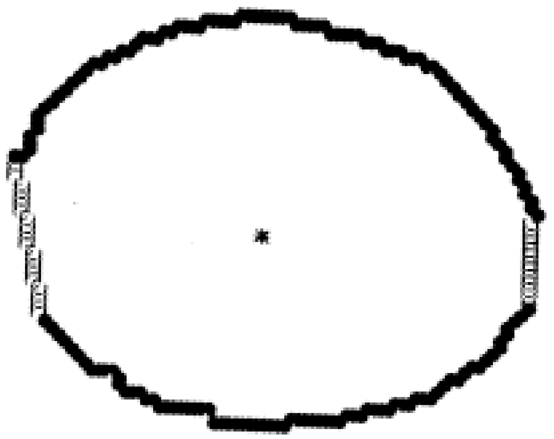
Figure 4-5. The best connected boundary computed for the center location shown in Figure 4.

evaluations, as illustrated by the lefthand center location.

In a map of boundary point locations about a center, each boundary point is represented by its x- and y-displacement from the center. Displacement values are used rather than pairs of angular position and radius because the displacement values are trivial to compute from the center location and the original edge map, whereas accurate angular position and radius values are expensive to compute. The computations that need to be done efficiently from this data structure are:

- compute the approximate radius for a boundary point from its x- and y-

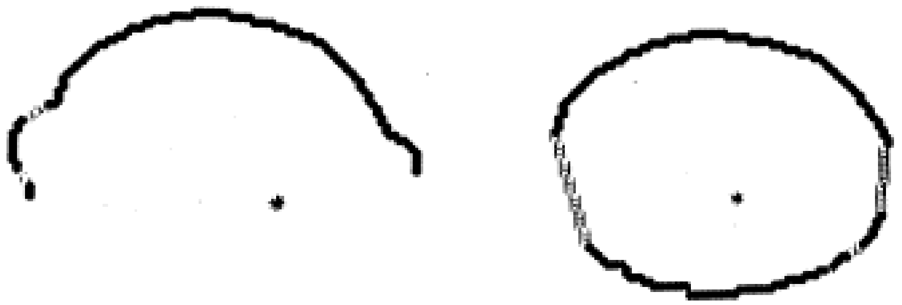

Figure 4-6. The best boundaries for two other center locations in the lemon figure. The center locations are marked with stars. The boundary on the right was computed for a location near the perceived center of the figure and it is very similar to the boundary computed for the perceived center. The boundary on the left was computed for a location further away from the perceived center and has a much lower evaluation.

displacements;

- compute the approximate angular position of a boundary point about the center from its x- and y-displacements;
- compute the angular deviation between the normal to the boundary at a given point and the radius line from the center to that point;
- given two boundary point locations that are close in angular position (i.e. much less than 180 degrees), determine which point is clockwise of the other;
- given a boundary point location, retrieve nearby boundary point locations.

The angular deviation is the basic measure of rotational symmetry for a boundary point about this center location. Ability to retrieve nearby locations is needed in building up connected boundaries of symmetry regions with this center. Boundaries of LRS regions are constrained not to switch angular direction. Since it is prohibitively expensive to compute accurate values for the angular position of a boundary position, relative angular positions computed from absolute angular positions are not reliable for points close together. Therefore, a separate operation is needed to check relative angular position during the process of building up connected boundaries. The approximate radius and angular position values are used for describing and evaluating regions with this center.

All of these computations can be done extremely efficiently. Calculating accurate radii and angular positions from displacements is computationally expensive. However, radius and angular position for any displacement values can be estimated using precomputed tables for locations with small displacements (currently up to 20 units of displacement in either dimension). Absolute angular position does not need to be particularly accurate, since there is a separate operation to determine relative angular position. The relative angular locations of two points can be computed quickly by directly computing the sine of the angular distance between them. If the x- and y-displacements of the points a and b are a_x, a_y, b_x and b_y, then $b_x a_y - a_x b_y$ is the sine of the angle from a to b, times the radii from the center to both points (i.e. the magnitude of the cross product of the vectors from the origin to a and b). The sign of this quantity indicates which point is clockwise of the other, assuming that the points are not close to 180 degrees apart. (There may be similarly efficient ways for determining relative radius for points near each other.)

The majority of center locations considered were not near the centers of regions at the previous coarser scale, so the only boundary point locations being considered are at locations within the fixed exhaustive search radius. A small number of center locations are the centers of coarser-scale regions and have boundary point locations at arbitrarily large distances from the center. The data structure used for this computation must allow efficient storage of boundary points for centers with only local boundary locations, while also allowing storage of boundary points further off for those centers that have extended computations. For any center, it must be possible to quickly retrieve locations adjacent to a given boundary point location, in order to be able to efficiently track connected boundaries. In the current implementation, the map of boundary points is stored in an array plus a list of points outside the bounds of the array. Initially, the array size is set so that the array can hold exactly the points within the current exhaustive search radius.[1] When a map is computed for a center with many points beyond this radius, i.e. when the program found a region with this center at a coarser scale, the size of the array is adjusted so that only few points lie outside the array.

A Local Rotational Symmetry region computed from a deviation map is represented by a center location and an ordered list of boundary point locations going counter-clockwise around the center. This list of points includes any points hypothesized to join disconnected sections of boundary and thus points included in the boundary are marked as "real" or "hypothesized". Since these connected

[1] The current implementation uses a square search area, rather than a round one, for programming convenience.

boundaries are computed from the deviation maps, the boundary points are represented by x- and y-displacements from the center. From this representation, symbolic descriptions of the sort described in Chapter 3 could easily be computed. In fact, some of the required computation, e.g. computing the range of angles covered, must already done in order to evaluate how good a proposed region is.

4.4. Evaluating regions

To form rotational symmetry regions from boundary points in a deviation map, we need to gather connected curves of boundary points that form regions that are as good as possible. The first issue involved in finding optimal regions is to come up with an exact definition of how regions are to be evaluated. As described in Chapter 3, evaluation of a connected boundary around a center involves the following factors:

- D: the average of the (unsigned) distances between the normals at points along the boundary and the corresponding radii;
- A: the angular length of the boundary, in percent of a circle;
- Classification of whether the boundary is open, closed, or spirals.

The angular length and the average deviation trade off against each other, so that a short region with extremely small deviation and a longer region with larger deviation are about equally acceptable. Experimenting with various ways of combining these factors suggested that the two factors should be related multiplicatively, in order to encourage regions which were good on both criteria, and that the average deviation should be weighted somewhat less heavily than the angular length. In order to keep spirals from having extremely high evaluations, I bounded angular length for open boundaries at 90 percent of a full circle, so a spiral is evaluated as if its length is at most 324 degrees, even if it is really much longer.

Since the boundary of what is perceived as one round region may be broken up into disconnected sections of boundary, e.g. by occlusions or attachments, the process of gathering connected curves may involve hypothesizing new boundary points to fill gaps in a curve. Therefore, another factor in evaluating a proposed region is fidelity to the input data. As an estimate of fidelity, I use:

- R: the percentage of points in the curve that are real boundary points found by the edge finder.

The current evaluation function combines this factor additively with the angular length and deviation term. For a closed boundary, the evaluation is:

- $\frac{100}{\sqrt{D}} + \frac{R}{50}$ (100 is the value of A, the percentage of the angular length of a full circle for any closed boundary!).

For an open boundary, the evaluation is:

- $\frac{min(90,A)}{\sqrt{D}} + \frac{R}{50}$.

The constant determining the balance between the evaluation term and the fidelity term was determined by experimentation. In order for a region to be considered minimally acceptable, its evaluation must be over a threshold, currently 7.0. Running the program on examples suggested that this was the approximate location of the cut-off between regions that seemed perceptually plausible and regions that didn't. Regions near this cut-off seemed marginal. It must also meet minimal requirements on angular length and deviation. Currently, the minimum angular length is 10 percent of a circle and the maximum average deviation is 20 degrees.

In fact, there are two ways in which a boundary point can be hypothetical, rather than a real boundary point of the original image. First, the original image might not have had a point at that location at all. Secondly, there might have been a point there, but not with the orientation assumed in the boundary. In other words, in the second case, the algorithm is assuming a boundary point where there was one in the image, but correcting the orientation to something it prefers. This happens frequently in the current version of the code.

In cases where the program corrects the orientation of a boundary point or in which it hypothesizes a new boundary point at a location very close to a real boundary point, it is not obvious how to interpret "fidelity to the input data". One obvious way to measure fidelity would be to say that such points are worse than real boundary points, but better than boundary points created out of whole cloth. Alternatively, one could say that since the program is so close to using a real boundary point, it should be encouraged to go the whole way and use the real point unaltered. This view amounts to saying that hypothesized boundary points are most acceptable when they are *not* near real points. My impression from looking at the program's results is that this second method of evaluating hypothetical points is perceptually correct, but this is a matter which needs more detailled investigation. The current implementation treats all hypothetical boundary points as equally bad, without taking account of closeness to real points.

Finally, the average deviation and the percentage of real versus hypothetical points were computed relative to the length of the boundary curve. It would also have been possible to compute them relative to the angle spanned by the boundary curve. The difference between these two formulations will be most obvious for regions whose boundaries spiral, so that the relationship between boundary length and angular length changes drastically between different sections of the boundary.

Refining the methods of evaluating possible regions so that they match human judgements is a matter for further research. Comparing the results of the current implementation, presented in Section 4.7, to my own intuitions suggests that the current evaluation function does not weight fidelity to data high enough. That is, the program seems to have more of a preference than I do for "correcting" the boundaries using hypothesized points, rather than following the input boundaries. Also, in looking at the program's analyses of oval or ellipse-type shapes, it seems as if the relative balance between angular length and average deviation is not set exactly right. The program analyzes some shapes as one round region when I consider them on the borderline between a long region with round ends and one round region. In this case, the behavior of the program is qualitatively correct, but the point at which it switches from one type of analysis to the other is wrong. The location of this change-over point is determined by the relative balance of the angular length and average deviation terms in the evaluation function. Detailed study of human perceptions of these shapes would be necessary to make the program's evaluations exactly match human judgements.

4.5. Building connected regions

Given a function for evaluating how good a region is, the next step is to devise an algorithm to build optimal connected region boundaries from the points in a deviation map. The algorithm that I will describe is the one implemented for this thesis. This algorithm does a reasonable job of producing optimal regions, but the technique it uses is only a heuristic. It is also likely that mathematical analysis of the problem of creating regions and finding locally optimal ones will result in more elegant, provably correct, possibly more efficient algorithms for computing symmetry regions. In particular, it seems that the regularization methods discussed by Torre and Poggio (1984) or the optimization methods discussed by Blake (1983a, 1983b) that have been used to solve problems in low-level vision might be applicable to the LRS optimization problem. Alternatively, it may be possible to directly construct a provably optimal algorithm operating on the discrete set of oriented edge locations. This is a topic for future research.

In order to explain the implemented algorithm, suppose, for the moment, that it was possible to use some fixed threshold on deviation from normal to decide which boundary points should be incorporated into the connected regions. In this case, an algorithm doing almost the right thing would first gather maximal connected boundaries from the points better than the threshold, and then attempt to join the ends of these boundaries into boundaries that are as long as possible. As described in Chapter 3, there are a variety of constraints on possible joins between two pieces of boundary:

- The points hypothesized in joining the two pieces of boundary should have deviations better than the deviation threshold, in order to maintain the constraint that the deviations of a region are better than the threshold;
- It is only plausible to join two boundaries if the angular distance of the join is below about 40-50 degrees;
- If the piece of boundary hypothesized in the join is long compared to the pieces of boundary being joined, the percentage of the resulting boundary that is real will be low enough to negate the advantages of joining the boundaries;
- The angular length of the join should be small compared to the lengths of the boundaries. (In the current implementation, it must be less than the longer of the two boundaries and less than three times the length of the shorter.)

Given these constraints on when two pieces of boundary can be joined, the current implementation uses a greedy algorithm to produce the best boundaries for a fixed deviation threshold. First, maximal connected boundaries are extracted.[2] Then, the algorithm starts with the longest boundary and tries to join other sections of boundary to it, starting with the boundaries whose ends are nearest. It should be noted that in order to close boundaries when appropriate, one of the candidates for joining is the other end of the longest boundary. Currently, ends

[2] All these boundaries go counter-clockwise around the center of the region.

are joined with a straight line segment, an approximation which is relatively crude for large gaps. When a plausible join is found, the two boundaries are merged and the process is repeated until either the longest boundary is closed or it cannot be joined to anything else, at which point it is removed and the process is repeated on the remaining set of boundaries. This algorithm will miss some possibilities for creating boundaries. For example, the overall optimal boundary for a set of points may involve joining boundaries whose ends are not locally the best pair to join and it may be made up of sections of boundary which are smaller than the maximum connected sets of boundary points.

In fact, it is not possible to impose a fixed threshold on angular deviations. As described in Chapter 2, the optimal boundary for a center may involve allowing in some points with bad deviations in order to be able to join the points with low deviations into long boundaries. Thus, my program iterates the above algorithm for finding regions over a series of thresholds on the angular deviation. (The current implementation steps from 20 degrees to 50 degrees by 5 degree increments.) The candidate regions for a given center are all of the regions for all choices for the deviation threshold. Again, this algorithm is not guaranteed to find the optimal region for a given center, although empirically it does a reasonable job. A further problem with the algorithm is that it seems to be doing the same work more than once. Specifically, when an optimal set of regions have been found using only points meeting a restrictive threshold on angular deviation, it seems that it should be possible to *extend* this solution when the deviation threshold is relaxed, rather than computing the regions for this new threshold entirely from scratch.

Trying to develop an efficient algorithm for computing provably optimal connected regions is a topic of current research. As discussed in Section 3.6, the current algorithm uses two distinct measures of fidelity to the input data: the percentage of real boundary points in the region, which is a global measure of fidelity; and the length of the join between two sections of boundary relative to the lengths of the two sections of boundary, a local measure of fidelity. It may be the case that the local measure is sufficient to account for the perceptual data. Note also that joins are restricted to 40-50 degrees in length. These two facts together suggest that the process of creating connected boundaries may be restricted to considering only a *local window* of the map of boundary points when it is creating optimal joins. Such a restriction might make it easier to develop direct optimal algorithms for building connected boundaries. Furthermore, this locality might account for the preference for closed boundaries: if the algorithm only looks at a small area around the gap between two long sections of boundary, it cannot tell whether the far ends of these pieces of boundary are connected or not. Thus, the same procedure that connects two disjoint sections of boundary would also connect two ends of the same boundary, because it cannot distinguish the two situations.

4.6. Non-maximum suppression

When candidate regions have been produced for the centers in an image, the locally optimal regions must be selected. As I pointed out in Chapter 2, regions computed for centers near the perceived center of a region may be relatively good, though worse than the best region for the perceived center. Further, for each center, we will in general generate a number of regions which are basically similar, except for small variations in choice of boundary points and joins. Therefore, it is necessary to suppress regions which are basically similar to a better region.

The important issue in suppressing sub-optimal regions is determining when to consider two regions "basically similar". It is not obvious what the perceptually correct definition is and I will discuss the matter in more detail in Section 5.3. The current implementation considers a region similar to a better region when more than 50 percent of its boundary points are the same as the boundary points of the better region. Note that this definition is *not symmetrical* in its two arguments. Thus, a large region and a better small region, may co-exist even if they overlap. An example of this situation would be an analysis of an oval as one large round region co-existing with analyses of its ends as being more coherent half-round regions. The algorithm iteratively selects the best region from all regions generated and removes it and all regions similar to it from the list of regions. The end result is a pruned list of distinct optimal regions.

4.7. Examples of Output

This program has been run on over 30 images of objects or groups of objects. The examples shown in this section were chosen to illustrate typical behavior of the program on diverse types of shapes. They show examples it works well on, as well as examples of the types of errors that the current implementation makes. The reader should bear in mind that the input grey-scale images have not been displayed at full resolution and that such images reproduce poorly. Thus, the region boundaries shown are a better index of the input resolution than the grey-scale images. Except where explicitly mentioned, *all* LRS regions found by the program are shown, including regions generated by fine-scale clutter, reflections, and shadows. These analyses were all done with the same settings for all parameters of the algorithm.

The first example, shown in Figures 8 to 11, shows the full multi-scale analysis of the spanner wrench image shown in Figure 7. This example illustrates what the multi-scale LRS analysis of a shape looks like in full gory detail. Boundary points are drawn as open circles whose size is proportional to the amount the image has been smoothed. (This is a graphical device for making blurred boundaries look appropriately blurred.) When LRS regions are shown, the boundary of each LRS region is indicated with filled circles, points hypothesized in joins are indicated by large open circles, and the original boundary points are indicated by small open circles. The size of these points is also proportional to the amount of smoothing. Selected radii from the boundary of the LRS region to its center are also shown. As you can see, analyses vary more or less smoothly between adjacent scales, with occasional sharp changes as new detail, such as the cut-outs in the jaws of the wrench, becomes visible.

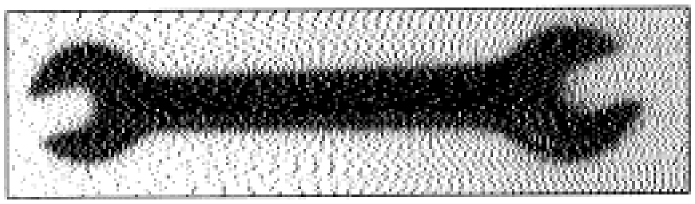

Figure 4-7. A grey-scale image of a spanner wrench

The rest of the examples will only be shown at a single scale. This is generally the finest scale to which the analysis was run, with the exception of a few figures whose boundaries were not detected by the edge finder at the finest scale. Figures 12, 14, and 16 show grey-scale images of figures with one round or oval region, of varying proportions. Figures 13, 15, and 17 show the LRS analyses of these figures. For figures which are clearly one round region or clearly a long region

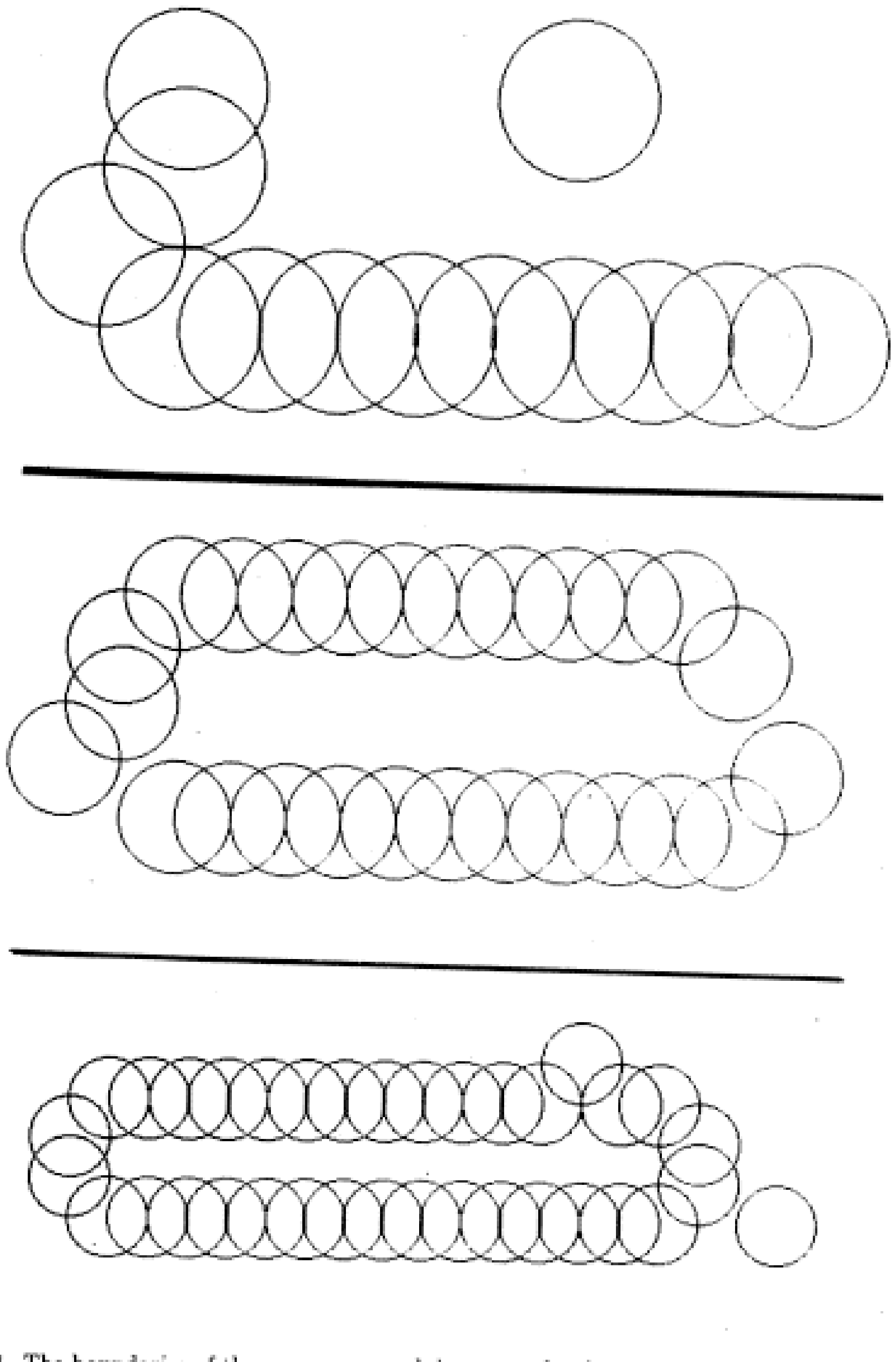

Figure 4-8. The boundaries of the spanner wrench image at the three coarsest scales. There were no good LRS regions found at these scales.

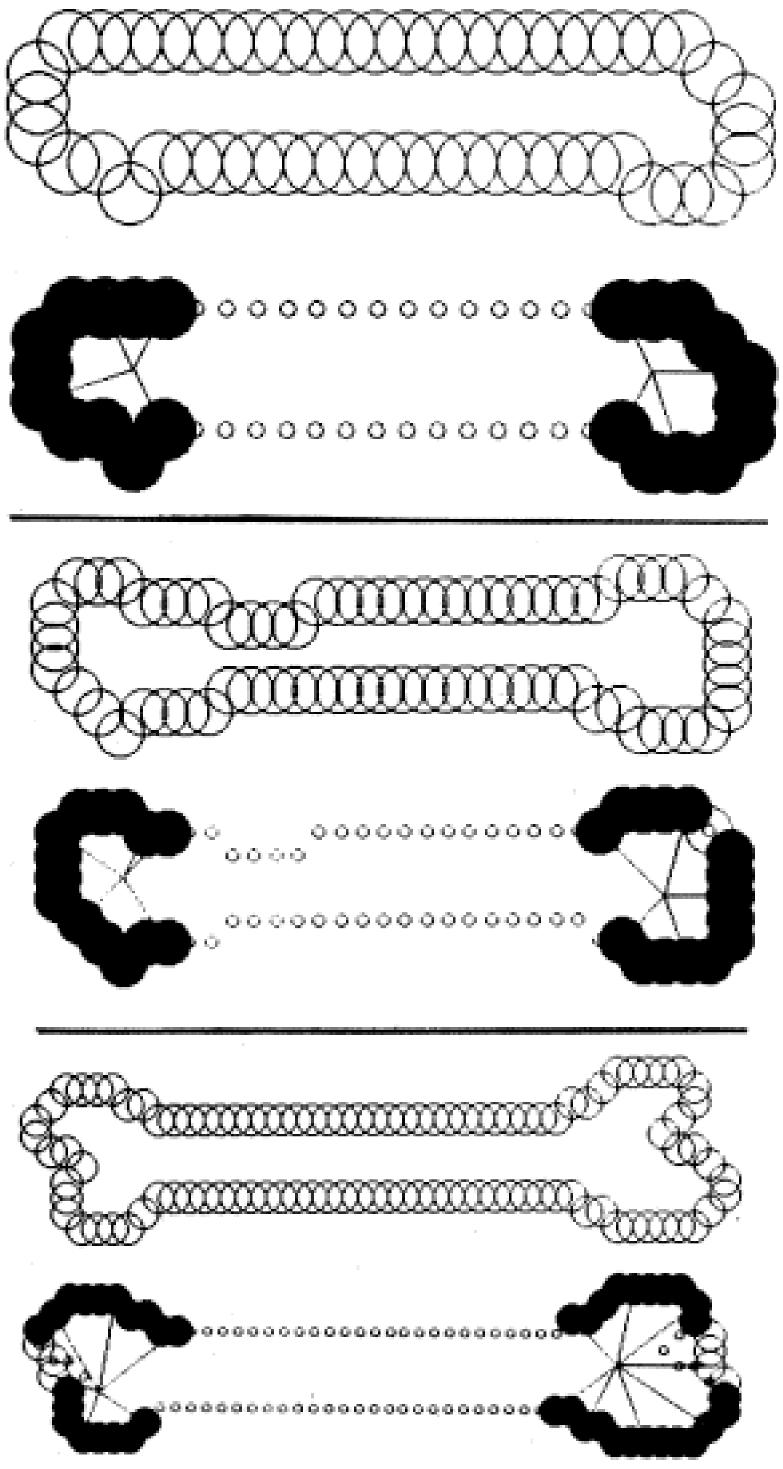

Figure 4-9. The boundaries and LRS regions of the spanner wrench at the next three scales. For each scale, the original boundary is shown, followed by the same boundary drawn in small circles with the boundaries of LRS regions indicated by larger circles. Selected radii from the boundary of the LRS region to its center are also shown. The size of circles representing boundary points is proportional to the amount the image has been smoothed.

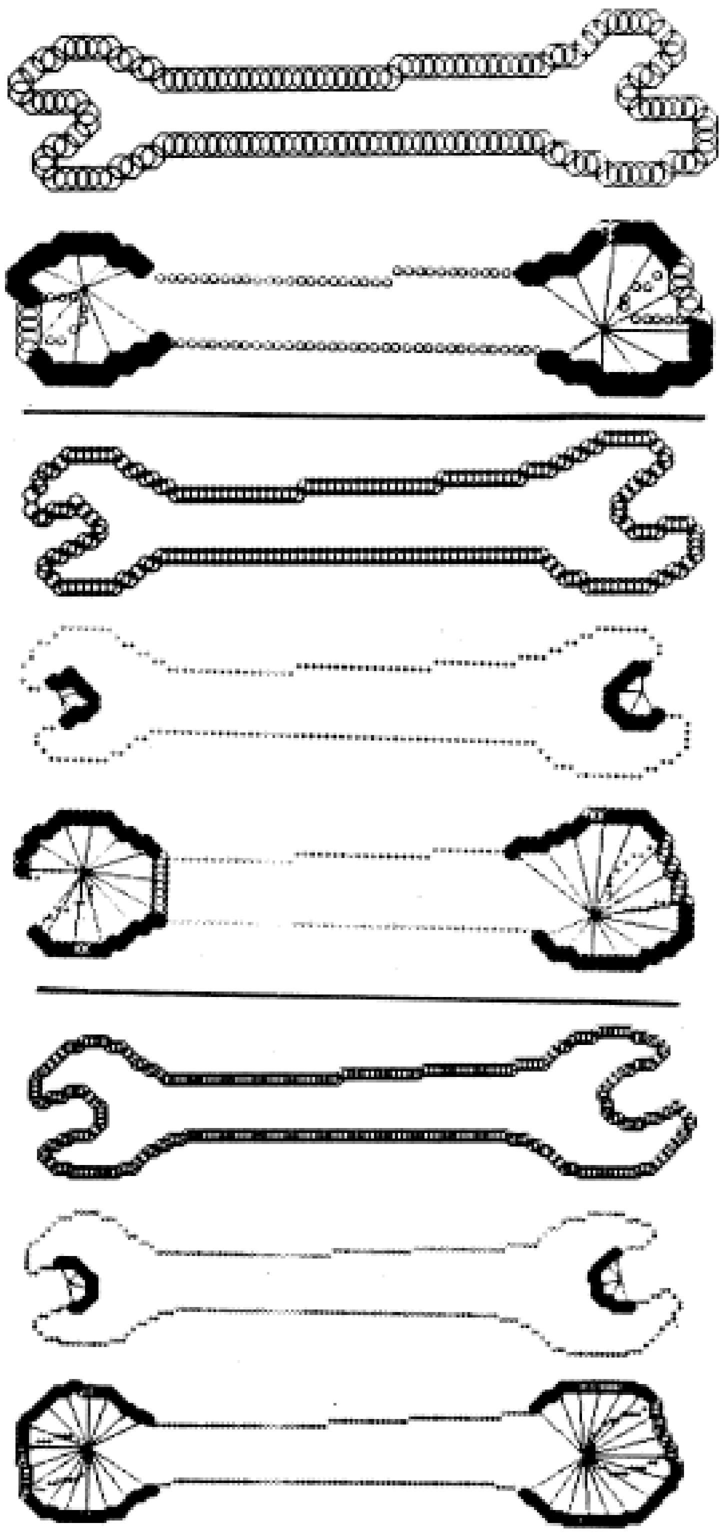

Figure 4-10. The boundaries and LRS regions of the spanner wrench at the next three scales. The cutouts of the wrench have started to be large enough and well-enough defined that they emerge as plausible LRS regions.

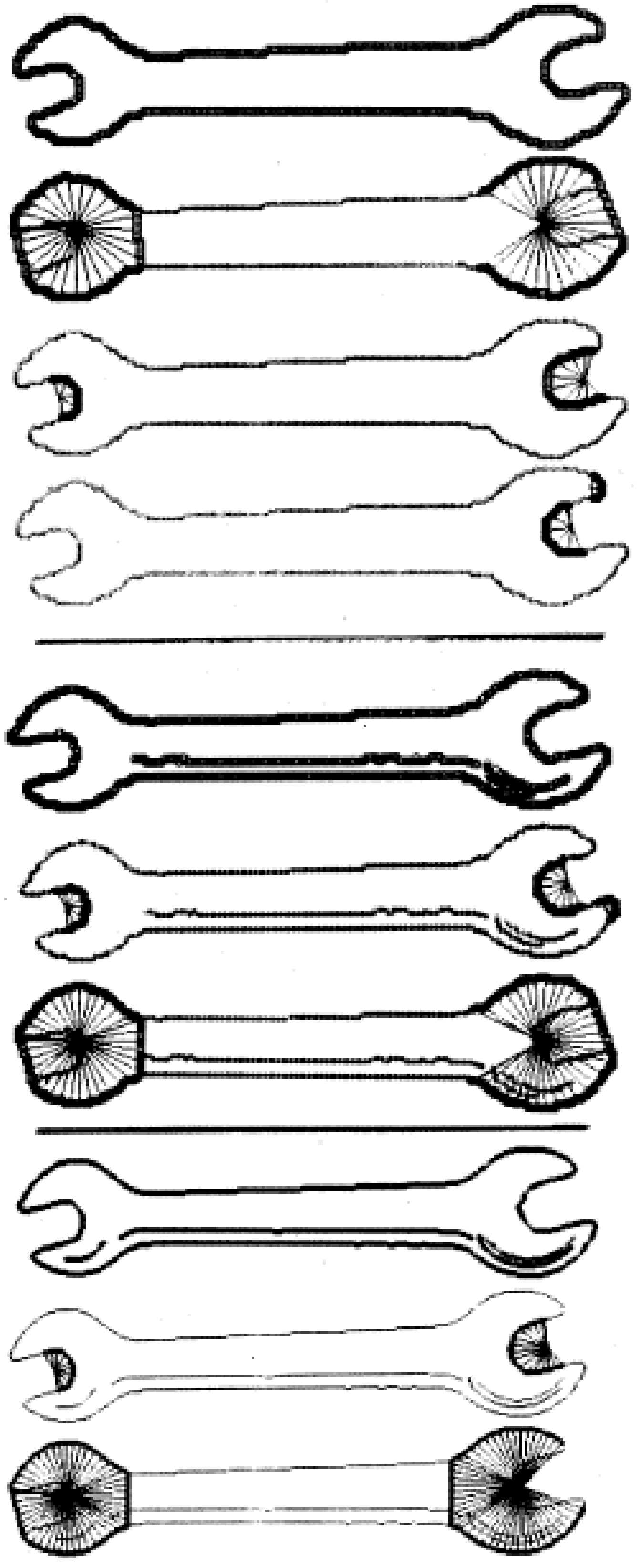

Figure 4-11. The boundaries and LRS regions of the spanner wrench at the finest three scales.

with two ends, the program finds the correct analysis robustly. For intermediate figures, the program finds one or both of the analyses, but in a somewhat degraded form, probably due to the roughness of the non-maximum suppression heuristic used. Furthermore, the point where the program makes the change in analysis does not quite agree with my perceptions: some of the figures it analyzes as clearly one region seem to me to be on the borderline, e.g. the pecan in Figures 16 and 17.

A second problem, illustrated most clearly by the hexagon image, is that the criterion for determining when one LRS region is a less good variation of another, 50 percent overlap in boundaries, is not sufficiently robust. When there are alternative parallel boundaries close together, or where there are many points added by joins, two regions that are perceptually "basically the same" can fail to have this many boundary points in common. Adding some measure of region overlap as well as boundary overlap would improve the robustness.

Figures 18 and 20 show images in which the round region has been broken up by attachments. Figures 19 and 21 show the LRS analysis of these shapes. The program proves to be able to detect round regions even when other regions are attached to them.

The next figure, Figure 22, shows grey-scale images of a coffee mug and a teapot. Figure 23 shows the LRS regions found for these figures. In general, the program finds the right regions. However, note the error in analyzing the handle of the teapot: the program creates a region by splicing together parts of two concentric circles. One problem with the current implementation of Local Rotational Symmetries is that the balance between fidelity to data and closeness to an exact symmetry is wrong. Figure 24, shows grey-scale images of a spiral, a key, a car part, and another spanner wrench. Figures 25 to 27 show summaries of the types of regions found in these figures. In figures in which there are sets of edges which look locally like parts of concentric circles with close radii, the program tends to "jump tracks," building counter-intuitive analyses because it is not giving high enough weight to following connected boundaries. It might seem as though considering the color of regions to the sides of the boundaries might help the program make this choice correctly. However, the program's analyses still seem counter-intuitive when one looks only at the boundaries of the figures rather than at the grey-scale images.

Two other problems shown up in the spanner wrench figure. First, the pro-

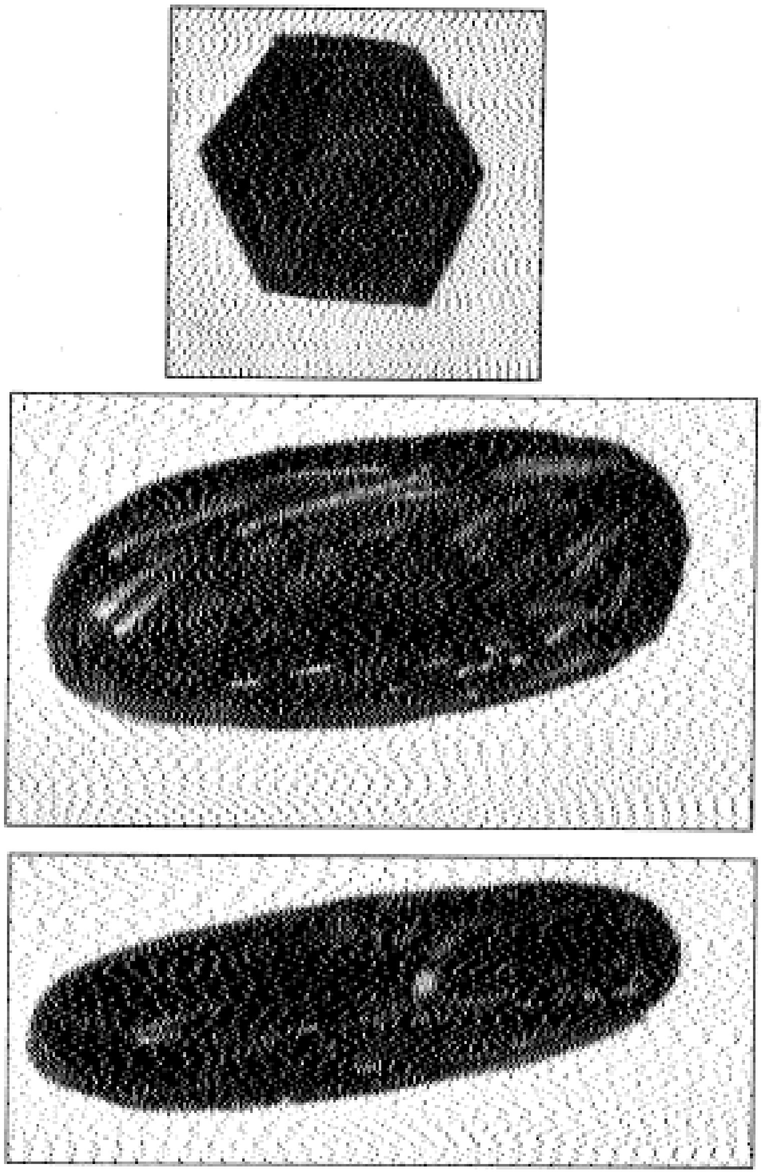

Figure 4-12. Grey-scale images of a hexagon and two ovals, drawn on a whiteboard.

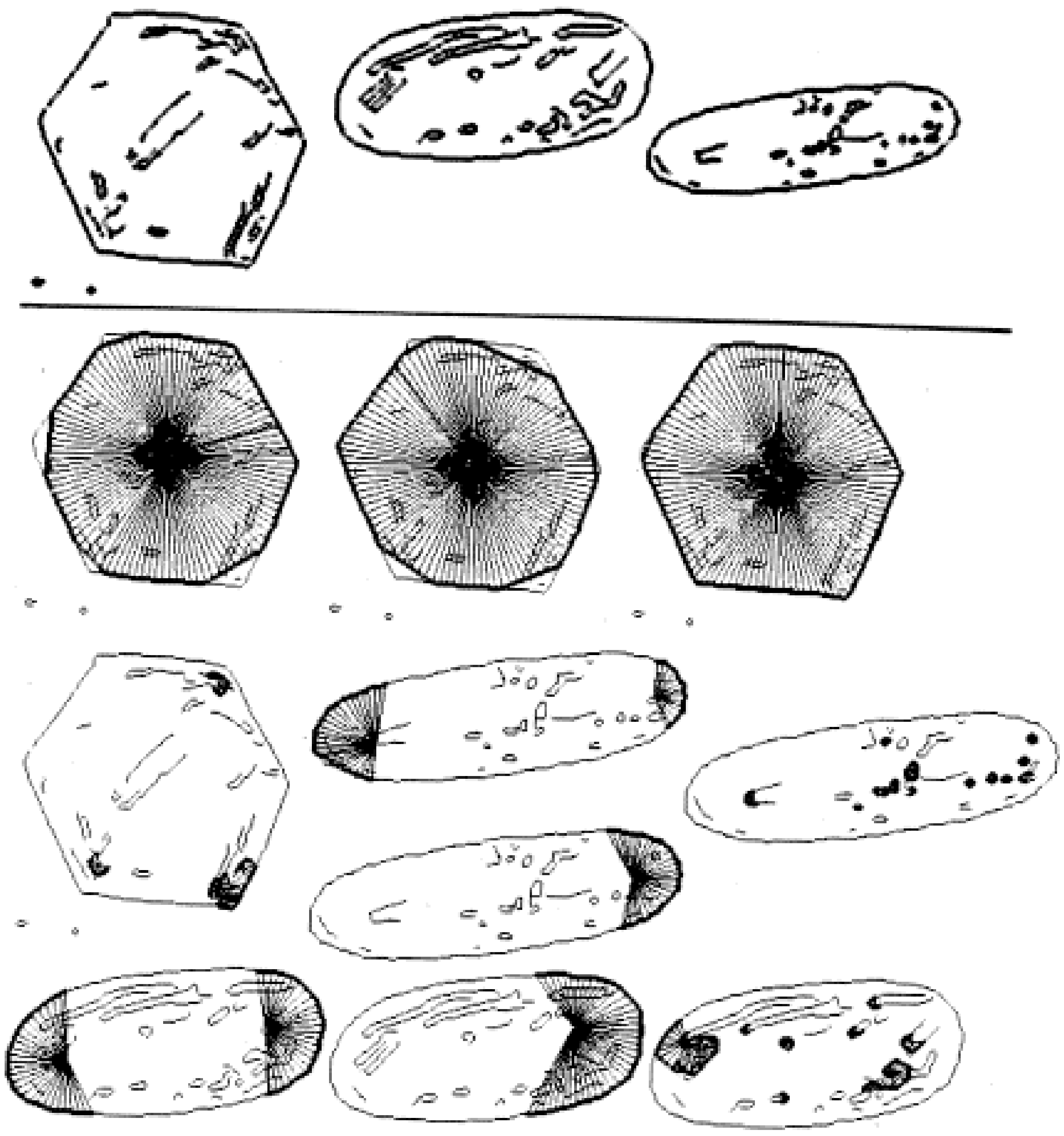

Figure 4-13. Analysis of the images from Figure 12. The boundaries found by the edge finder are shown in the top row. Below that are all the LRS regions found by the program for these figures.

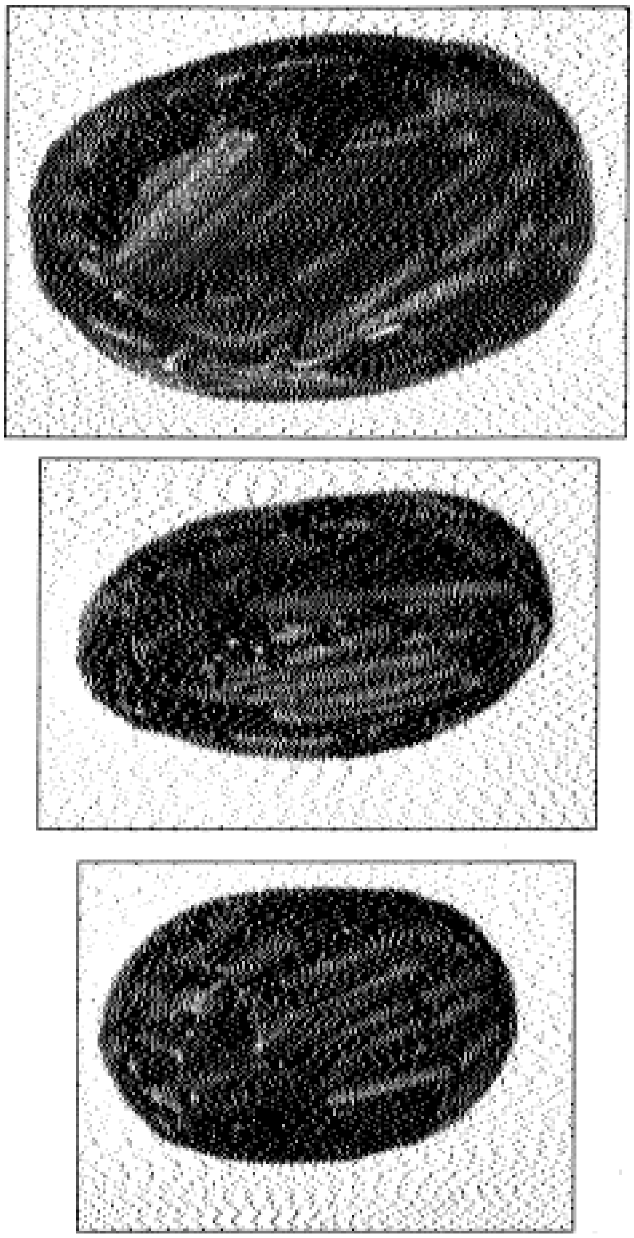

Figure 4-14. Grey-scale images of three ovals, drawn on a whiteboard.

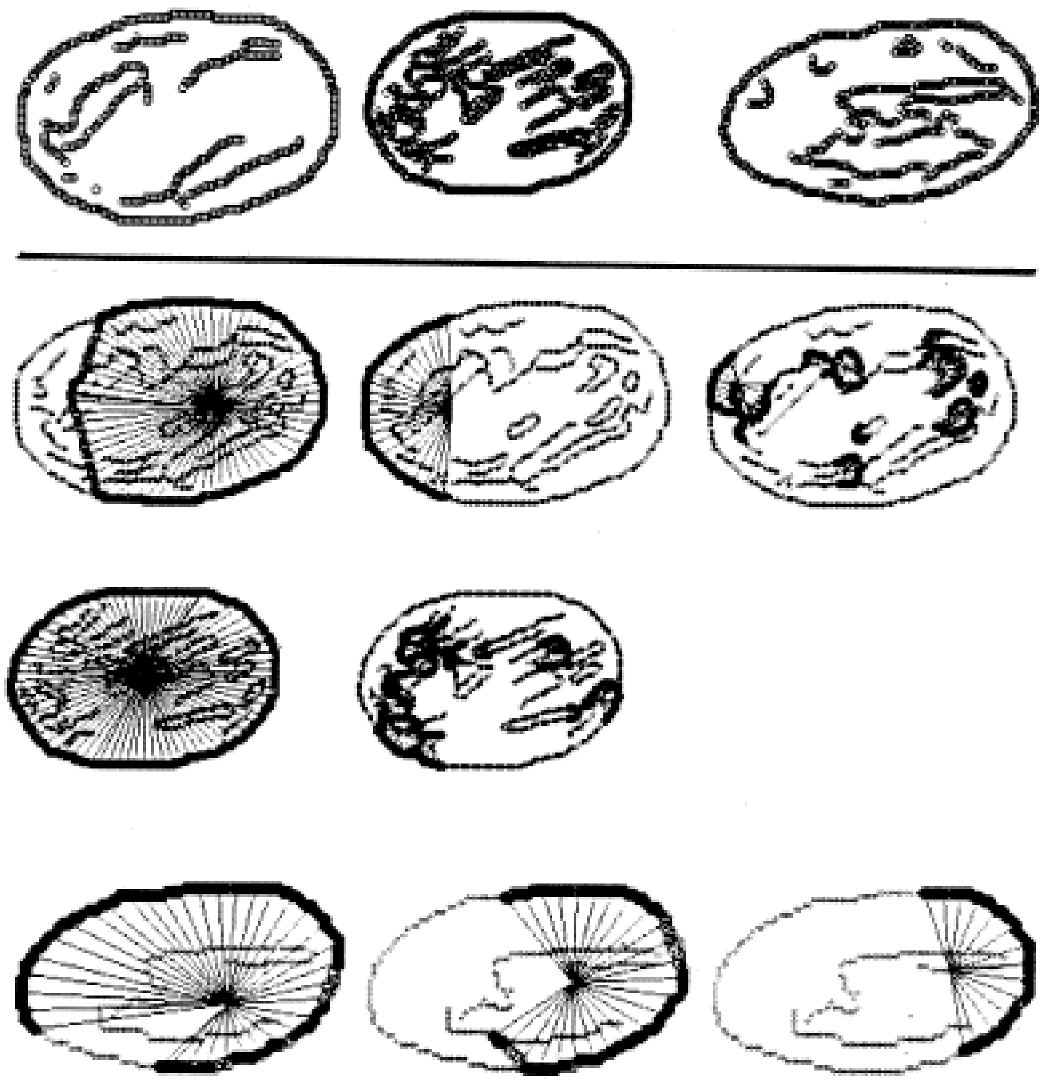

Figure 4-15. Analysis of the images from Figure 14.

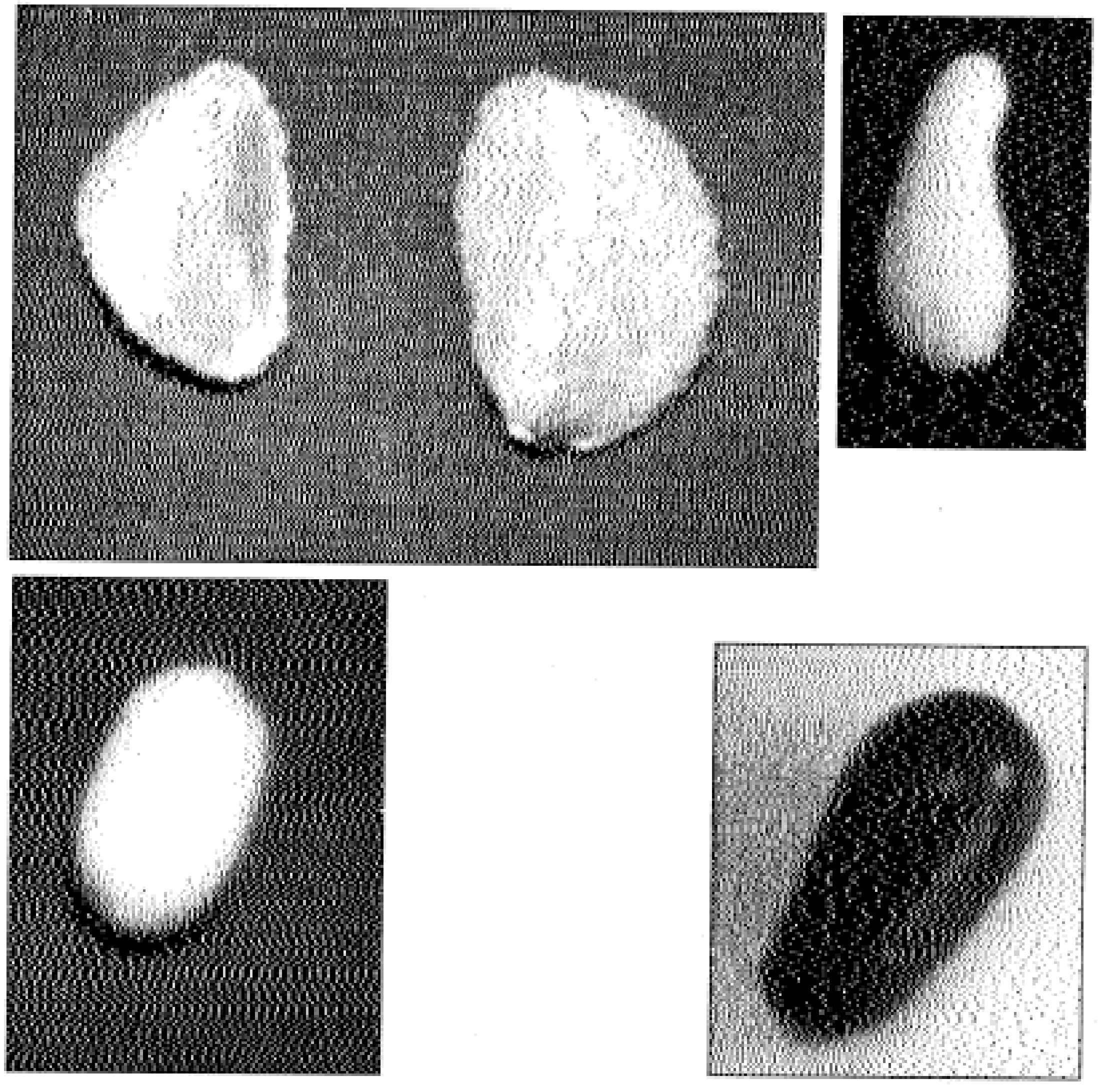

Figure 4-16. Grey-scale images of two brazil nuts (upper left), a squash (upper right), a pecan (lower left), and an eggplant (lower right).

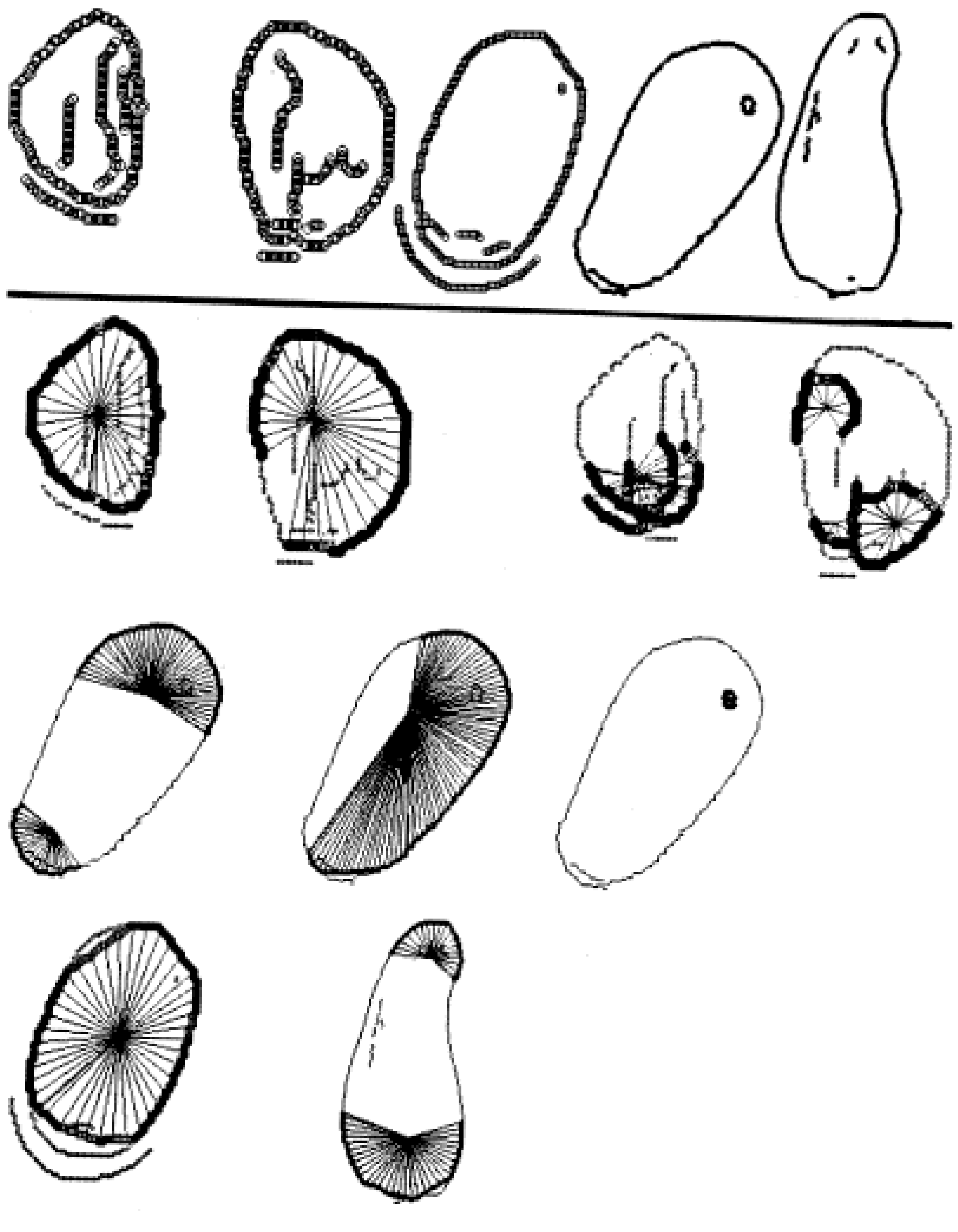

Figure 4-17. Analysis of the images from Figure 16.

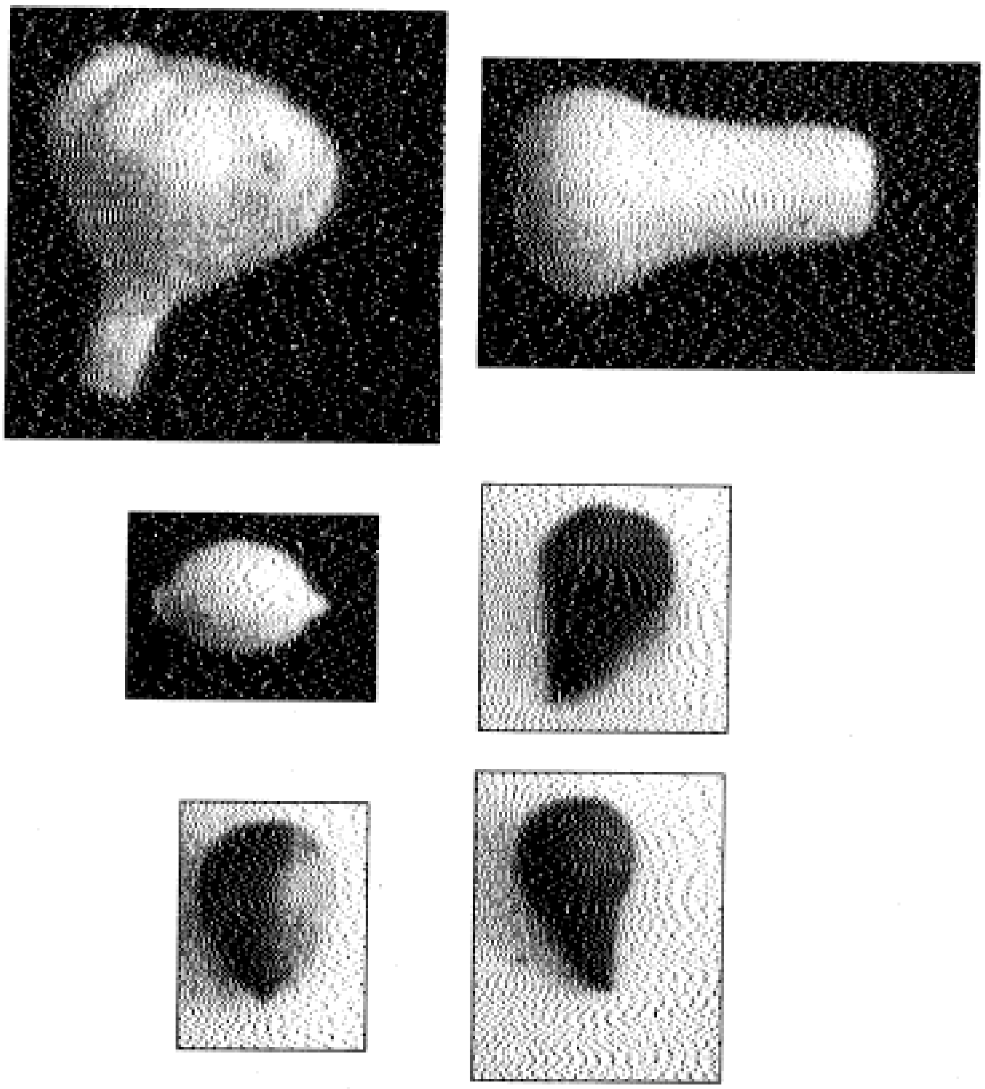

Figure 4-18. Grey-scale images of objects with attachments. Top row: a garlic bulb and a squash. Middle row: a lemon and a pear. Bottom row: two more pears.

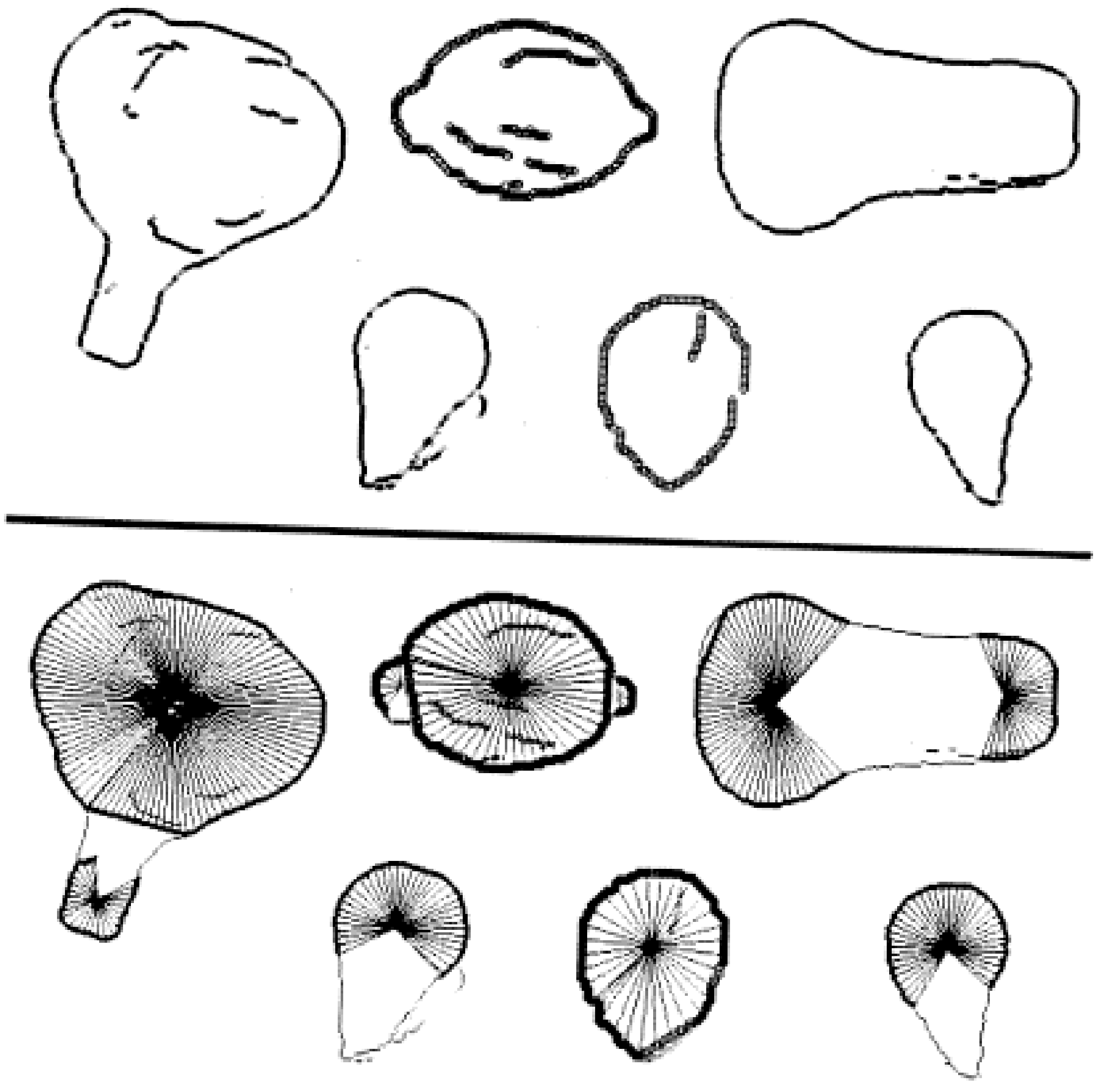

Figure 4-19. Analysis of the images from Figure 18.

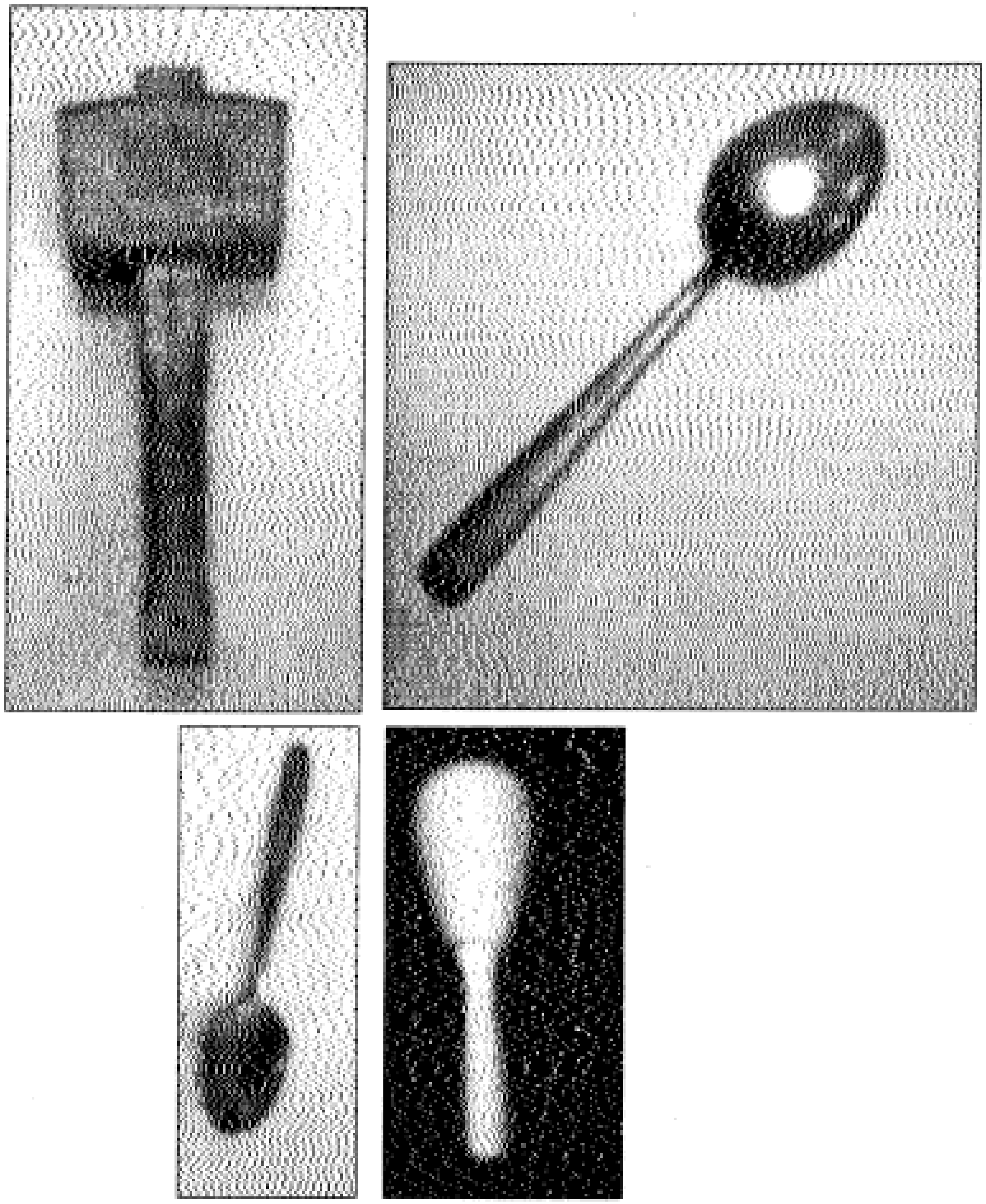

Figure 4-20. Grey-scale images of objects with attachments. Top row: a mallet and a spoon. Bottom row: another spoon and a darning egg.

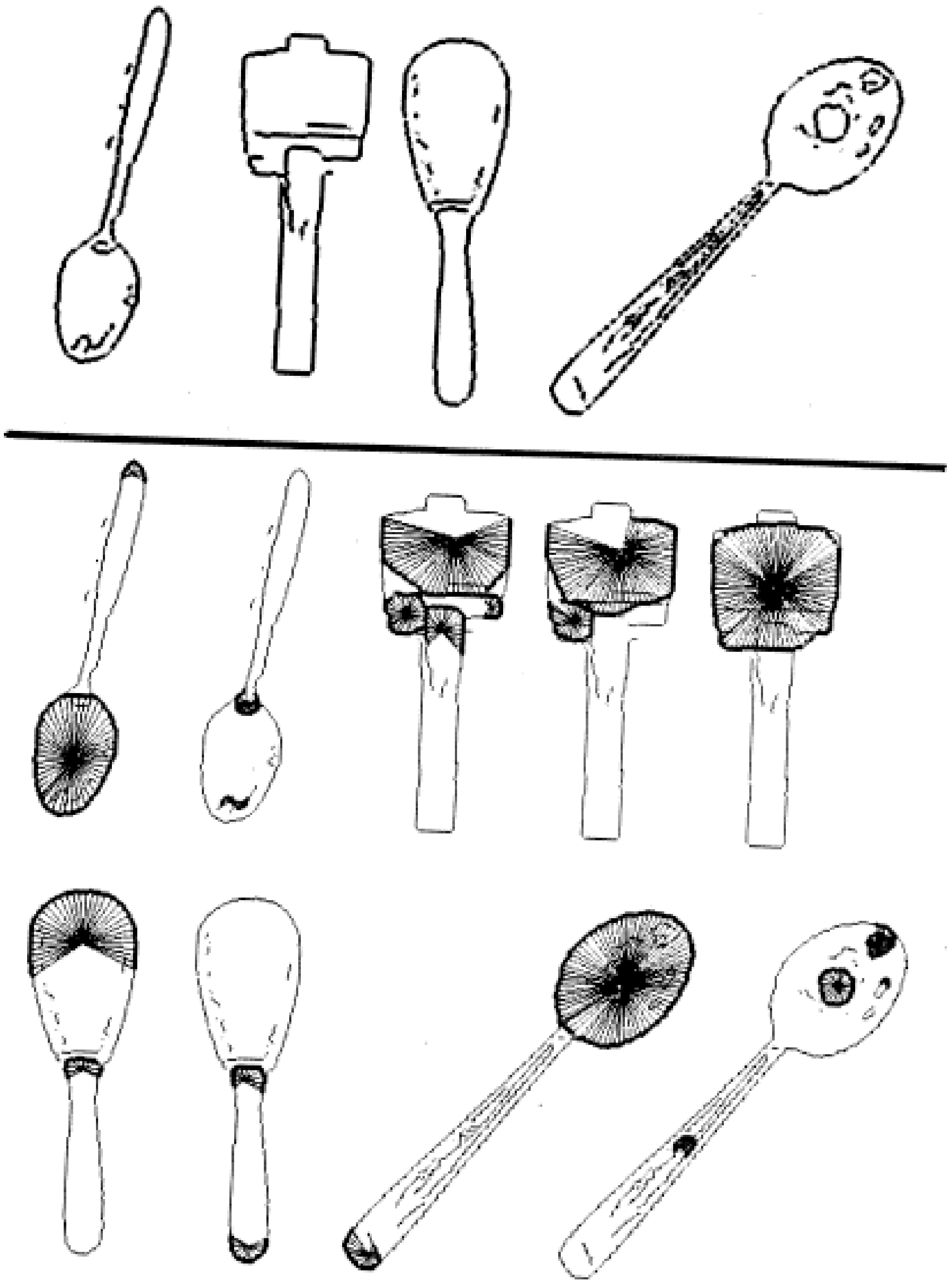

Figure 4-21. Analysis of the images from Figure 20. Note that the square end of the mallet handle was picked up as a partial round region in the analysis at the next coarser scale from the one shown. In that coarser scale, the corners of the end were slightly rounded.

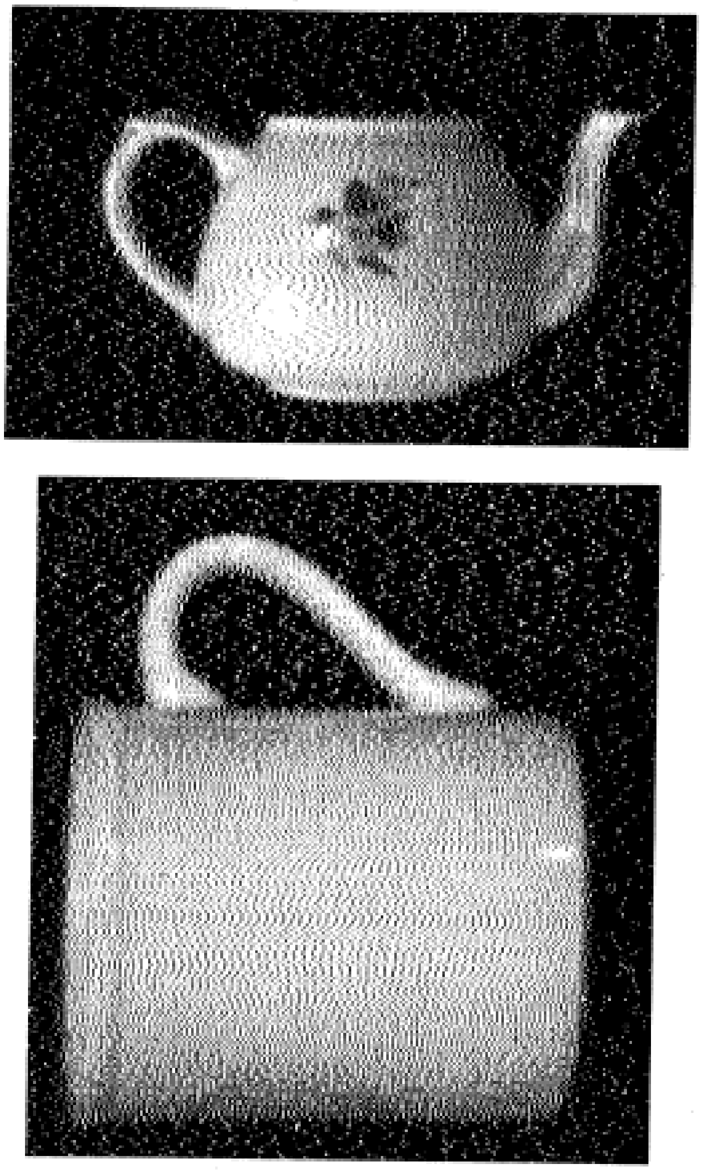

Figure 4-22. Grey-scale images of a teapot (top) and a coffee mug (bottom).

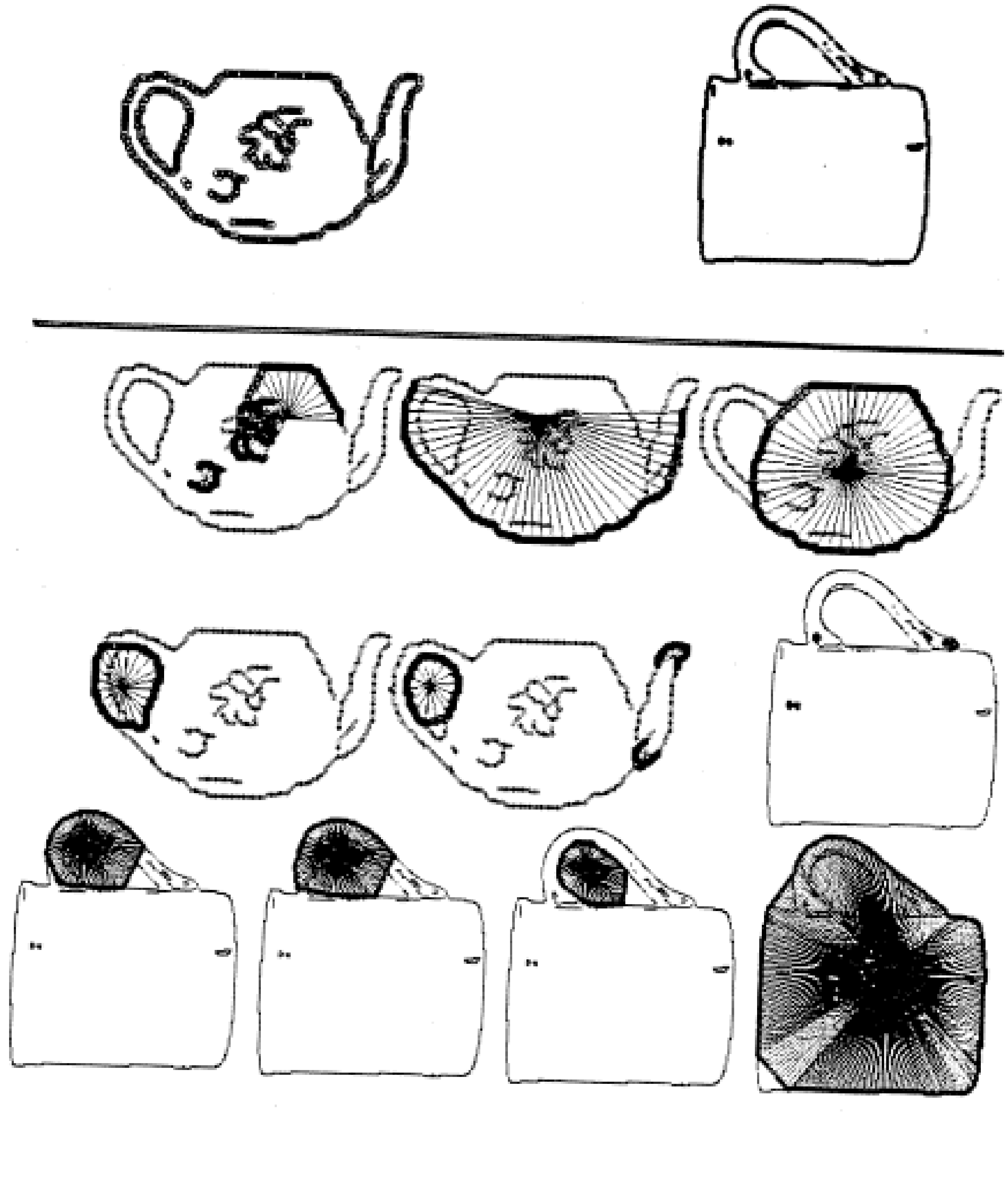

Figure 4-23. LRS analysis of the teapot and coffee mug images.

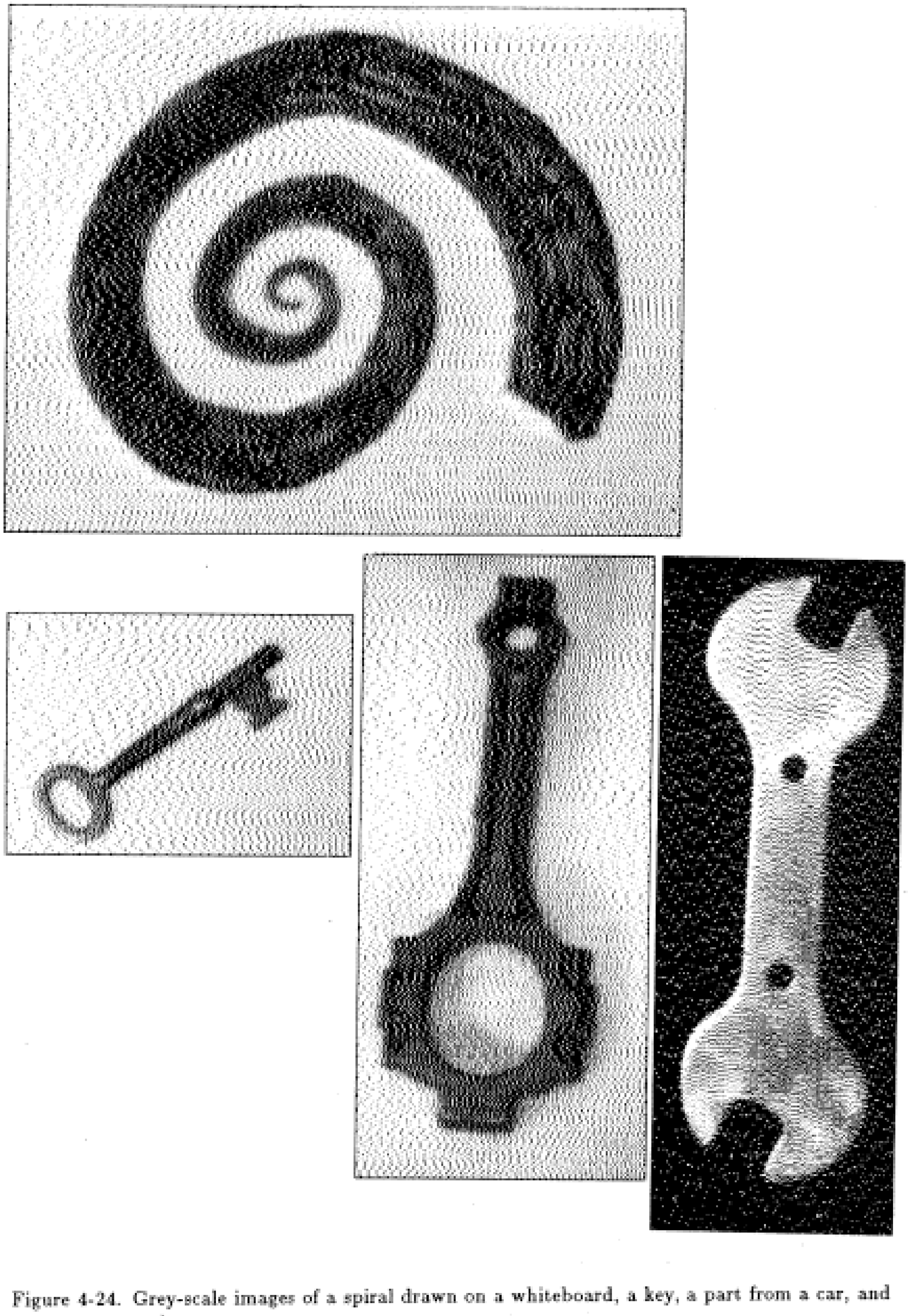

Figure 4-24. Grey-scale images of a spiral drawn on a whiteboard, a key, a part from a car, and a spanner wrench.

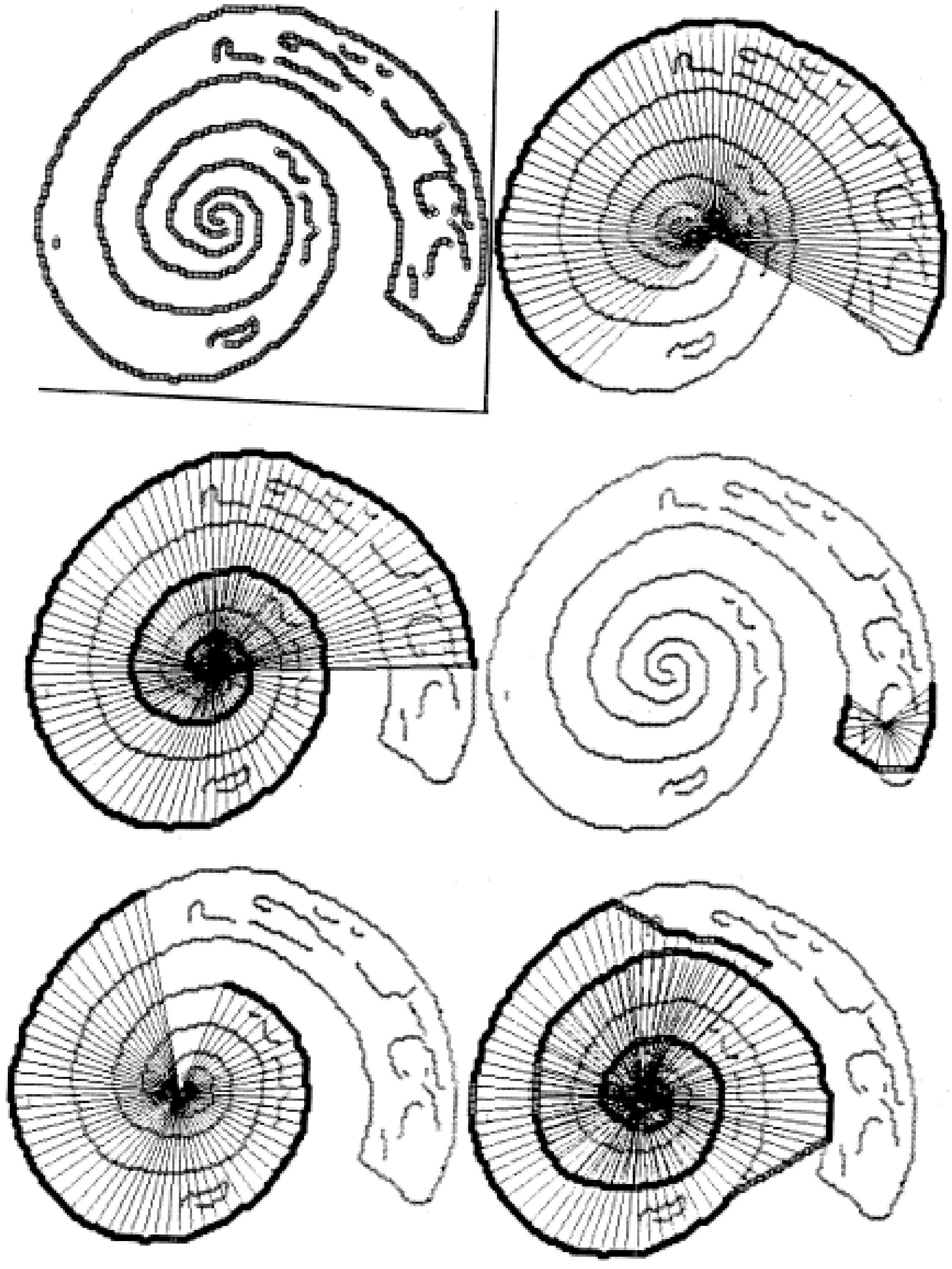

Figure 4-25. Sample regions from the LRS analysis of the spiral. The full analysis of the figure contained 7 different spiral regions (2 shown), 4 partial round regions (2 shown), 2 regions in the large end of the spiral (1 shown), and 4 regions due to noise.

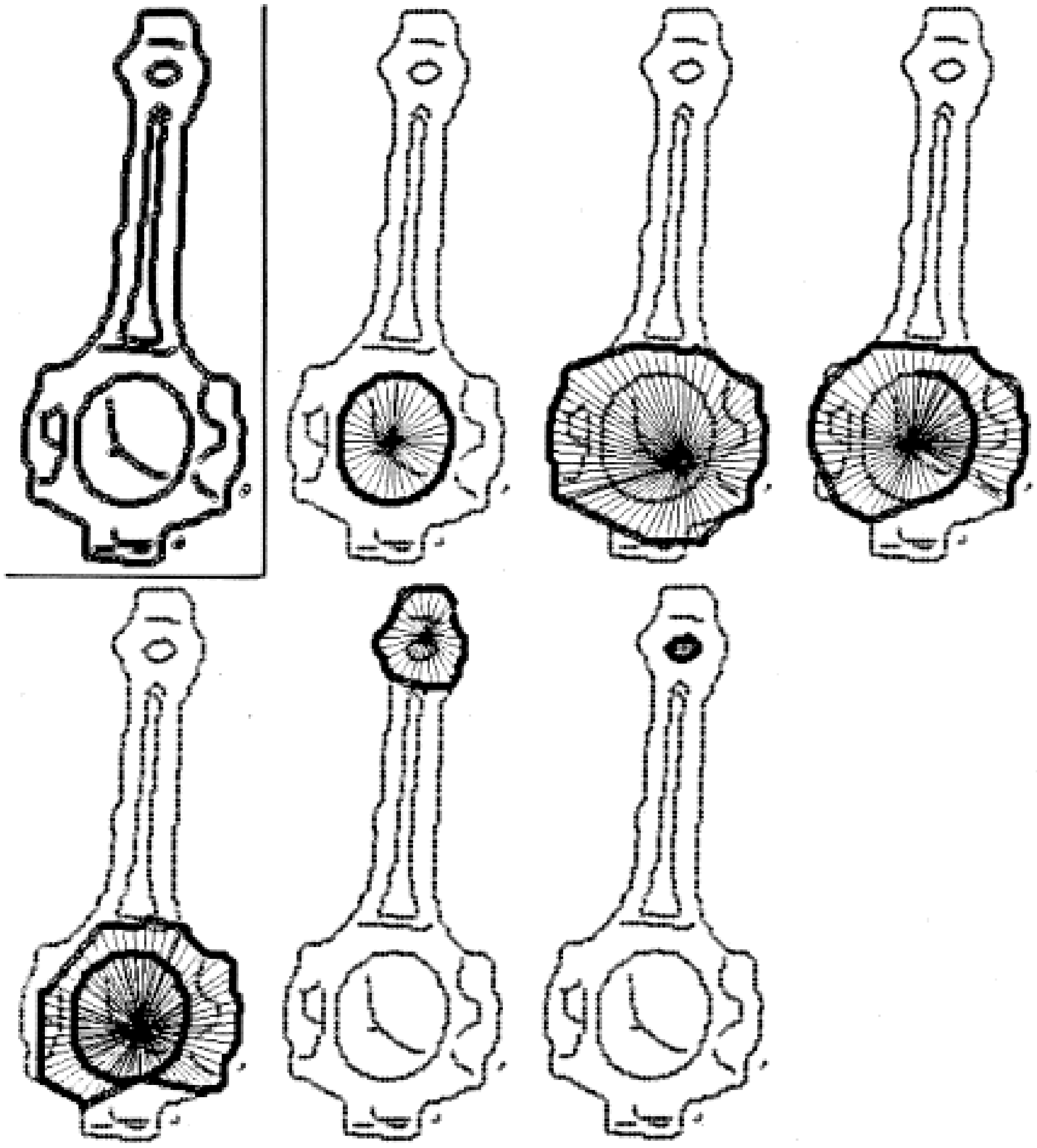

Figure 4-26. Sample regions from the LRS analysis of the carpart. The full analysis contained 1 region for the inside of the big end (shown), 5 alternatives for the outside (1 shown), 4 regions for the big end that spiral or cross themselves (2 shown), 2 alternatives for the outside of the small end (1 shown) 1 region inside it (shown), and 12 small regions due to clutter.

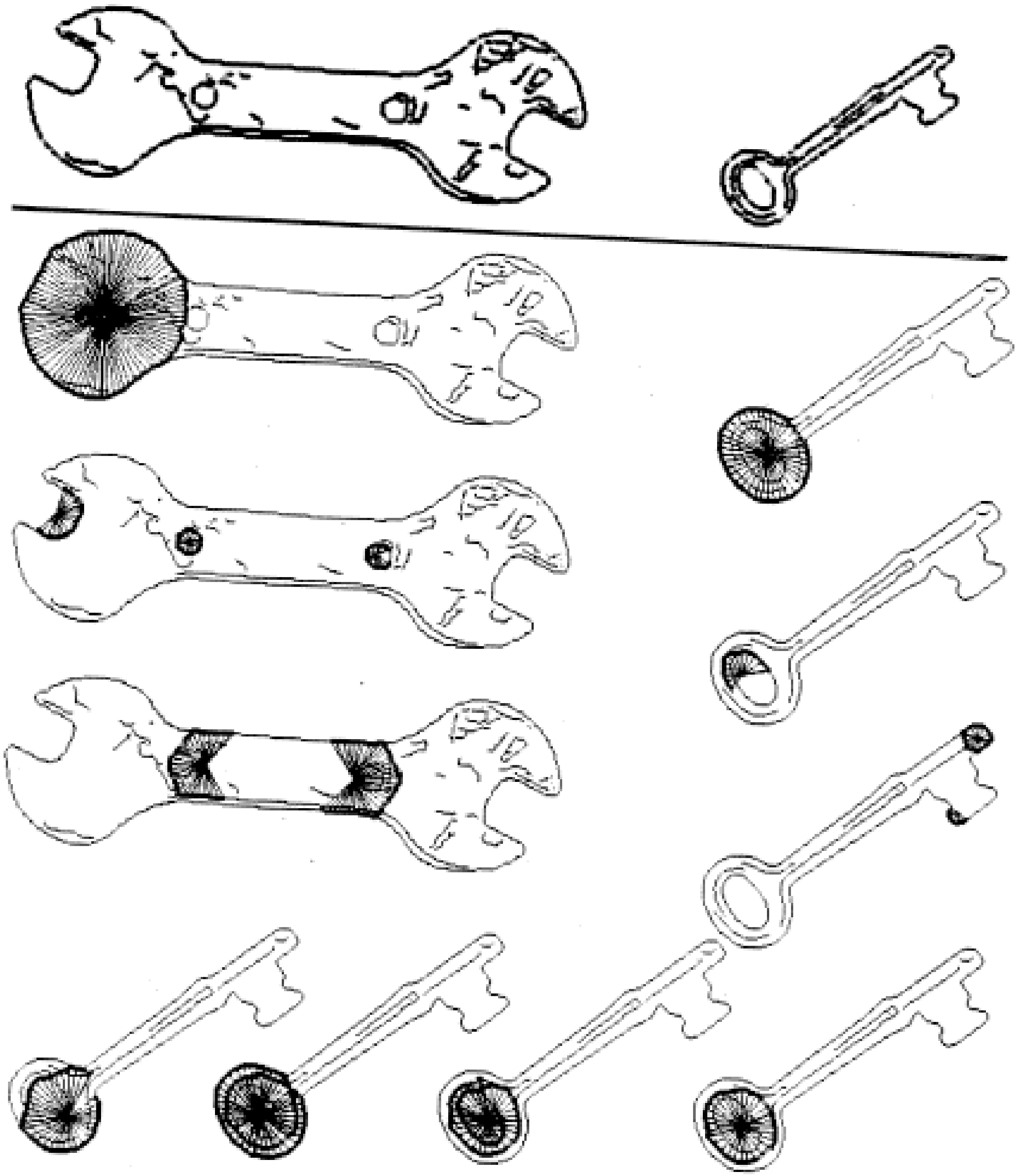

Figure 4-27. Sample regions from the LRS analyses of the spanner wrench and the key. The key analysis contained 2 regions at the tip of the key (shown), 2 other detail regions, and 11 variations on round and spiral analyses of the concentric arcs in the round end (6 shown). The spanner analysis contained 5 variations on the small holes in the handle (2 shown), the left end (shown), 2 variations on the left jaw (1 shown, variant is due to the shadow edge), 6 variations on the regions at the ends of the handle (2 shown), and 8 small detail or noise regions. The right end of the wrench was found at coarser scales but lost at this scale due to the shadow line. The right jaw was missed for unknown reasons.

gram builds counter-intuitive regions at the ends of the handle, by splicing an extremely short piece of contour together with two much longer pieces. There are other examples of similar behavior on other figures. What is wrong here is that the current implementation allows the gap in a join to be as long as three times the length of the smaller piece of boundary. From these examples, it seems that this threshold is too permissive. Secondly, although the program finds the right-hand head of the wrench at coarser scales, it gets confused by the shadow line that appears along the bottom of the wrench at finer scales. It first builds a spiral boundary for the right-hand head with a misplaced center location, and then (because the center passed down as a suggestion is bad), loses the region entirely. In cases like this wrench, it might be better to allow the region-growing algorithm to merge two boundaries which are very close together, when this would lead to a better analysis of the region.

In sum, the regions chosen by the program to describe input images are relatively close to what you or I would consider natural descriptions of the round regions in the image. Attachments, irregularities in region shape, and fine-scale clutter in the images do not prevent the program from finding these regions. There are also a variety of errors, most of which can be explained as reflecting slightly incorrect choices for parameter settings and the coarseness of the current algorithms for building regions and non-maximum suppression. The results are close enough to human perceptions to serve as a solid basis for doing more detailed investigation of human shape representation. More detailed psychological evidence is obviously necessary to refine the results.

4.8. Analysis of the algorithm

The running time of the algorithm can be broken down into two parts:

- the running time of the exhaustive computation
- the time spent pursuing fine-scale refinements of regions found at coarse scales

The exhaustive computation takes a constant (worst-case) amount of time per center location. Since the program only explores center locations within the exhaustive search radius of some boundary point, the total number of center locations is linear in the size of the input image. Thus, the running time of the exhaustive computation for one scale is linear in the area of the image at that scale (measured in pixels). Since the sum of the areas of all the smoothed images is a constant times the size of the input image (it is a geometric series), the running time for the entire exhaustive computation is linear in the size of the input. For similar reasons, smoothing the image and finding edges to make the input to the LRS algorithm also takes time linear in the size of the input.

Actually, this running time for the exhaustive computation is only strictly true as a worst-case estimate. In fact, the computation time spent per center is a function of the number and complexity of the boundaries around this center. Thus, the real expected running time of the exhaustive computation is also function of the "complexity" of the image at each scale. It would be nice if this measure of "complexity" correlated with psychological data, but I have no evidence either way.

Making finer-scale versions of regions found at coarse scales takes additional time which is a function of the number of good regions found in the image. The time spent computing refinements of a region gets larger as one moves to increasingly fine scales, because the number of points in the boundary gets larger. In fact, computing extremely fine refinements of a region that is stable over a large range of scales can take large amounts of computation. Therefore, it may be necessary to impose some limit on the maximum radius at which detailled refinement of a region is done.

The current implementation is slow on a Symbolics 3600. Analysis of a medium-size example shown in this section to the finest scale of resolution takes all night. However, most of the algorithm is highly parallel and would be speeded up drastically by parallel hardware or hardware assists. Since the computation for each center is independent, as much of it can be done in parallel as there is available hardware. Furthermore, it is likely that better theories of how to create optimal regions for a fixed center will decrease the computation time per center. Since the computation time is roughly linear in the area of the input, any such speed up in the time per center will result in a corresponding speed up in the overall running time.

Chapter 5: Issues in single-scale representation

In this chapter, I compare local symmetry representations to other types of representations for shape. I also discuss several issues involved in representing 2-dimensional shape at a fixed resolution: how to build descriptions of complex objects from the raw symmetry regions, how multiple representations for selected shapes can lead to a more stable overall shape representation, and how the color of regions may affect which symmetries within them are salient.

5.1. Alternatives to local symmetries

A wide range of different types of representations for two-dimensional shape have been proposed. There are three general classes of representations with potentially high coverage and descriptive power:

- Local symmetry representations;
- Generalizations of the Hough Transform;
- Model fitting techniques.

I will discuss these alternatives in detail. For a survey of other alternatives, see Ballard and Brown (1982) and Pavlidis (1977, 1982).

Local symmetry representations include Smoothed Local Symmetries, Local Rotational Symmetries, and the Symmetric Axis Transform (Blum 1973, Blum and Nagel 1978). The local symmetry represenations have several important properties:

- Shape models are local;
- Connectedness of boundaries or of axes is used to build regions out of local symmetries;
- Both regions and boundaries are represented;
- Axes and centers of regions are determined by the local symmetry construction.

The common idea behind these representations is that an input shape is analyzed by first searching for instances of shape models describing *local* relationships between sections of boundary. For instance, the model used in the Smoothed Local Symmetry representation is two sections of boundary that are locally reflections of one another. Representations of extended regions in the input shape, such as rectangles or hammer handles, are built up by joining instances of the local models to form connected boundaries and axes.

The Symmetric Axis Transform (SAT) is the ancestor of the other two local symmetry representations. This representation finds the centers of maximal circles that can fit within a shape and uses the centers of these circles as a description of the shape. The restriction that circles do not cross boundaries in a figure makes the representation very sensitive to small holes or internal color regions in a figure. If the no-crossing restriction is removed, the SAT becomes equivalent to finding centers which are exactly equidistant from two boundary points and normal to the boundary at these points. Although the SAT picks out reasonable pairs of corresponding points for elongated shapes, it assigns a perceptually incorrect symmetry center to them, as discussed in Brady (1983). For round shapes, the SAT picks out the correct center for a set of points with an exact local symmetry. However, since it requires the boundary points to be *exactly* normal to the center location and *exactly* equidistant from it, it only really handles regions which are exactly circular and it will be very sensitive to small deformations from this, as discussed in Section 3.8.

Local symmetry representations computed on fine-scale raw input boundaries are extremely sensitive to noise, as Agin (1972) pointed out for the SAT. However, this sensitivity can be removed by smoothing the image or the boundaries to remove noise, typically as part of a multiple-scale analysis (see Chapter 6). The shape primitives described by Crowley (1982) seem to pick out points similar to the SAT center points, but at multiple scales of resolution. However, it is not obvious how the two representations are related mathematically.

Generalizations of the Hough Transform have been described by Ballard (1981) and Davis (1982). These programs search an image for a parameterized class of shapes, e.g. circles, ellipses, or a fixed shape with rotation and scaling. One of the parameters will generally be location in the image. Each boundary point in the image votes for all combinations of parameters which would produce a figure containing that boundary point. Sets of parameters that receive large numbers of votes are taken to indicate the presence of an object of this class with that set of parameters (including a value for the location parameter). One problem with the Hough Transform is that it does not make use of connectivity information. As discussed in Chapter 3, connectivity information is crucial in determining what regions are perceptually reasonable. Brady (1983) discusses the same problem with the Hough Transform from a slightly different point of view. Secondly, the transform can only search for a limited class of shapes at one time. If the number of parameters is increased, the space of possible parameter combinations to be explored will get unmanageably large. Finally, the transform does not provide a theory of how to choose the reference point for a given shape.

Finally, there are a wide class of other shape description algorithms that search for instances of a class of shape models in an image. These algorithms differ as to what class of shapes they search for and what types of techniques they use to fit shapes to input data. The class of shape models that is best known in high-level vision work is Generalized Cylinders (Shani and Ballard 1984, Brooks 1981) which represent elongated 3-dimensional shapes, and their 2-dimensional analogs, called ribbons. As far as I know, the only class of shape models that has been proposed for describing round shapes is ellipses (including circles). Brooks, for example, uses ellipses to model round 2-dimensional projections of Generalized Cylinders. Sakaue and Takagi (1980, 1982) use iterative methods to fit circular models to data obscured by noise and occlusion.

These model-fitting techniques share two problems with the Hough Transform. First, they do not provide a good representation for irregular shapes. An irregular shape will be detected as a marginal match to a target shape and one is then left with the problem of describing the divergences. If the class of target shapes is increased, the computational cost of searching for them increases proportionately. In contrast, with local symmetry representations, a region is built up from local fragments that can be detected more or less independently. Thus, the class of possible shapes found is much wider: the radius function for an LRS region can be any smooth function (up to the current limits of resolution). But, because the region-finding process is more data-driven, local symmetries do not require searching a large parameter space.

A second problem with the model-fitting approaches is that they do not provide constructive definitions of the axis of an elongated shape or the center of a round shape. Axes are used in describing Generalized Cylinders. However, a given shape can have Generalized Cylinder representations based on more than one axis and the theory provides no way to decide which axis to pick for a given input shape. In contrast, the local symmetry constructions explicitly define an axis or center for a shape *constructively* in terms of the input shape. This means that local symmetry representations do not need an auxiliary algorithm for choosing axes, centers, or other types of coordinate systems. Brady (1982) also mentions that the axes of projections of Generalized Cylinders do not in general conform well to the perceived axes of shapes.

The difference between local symmetry representations and other types of shape models can be summarized by saying that local symmetry representations lend themselves to shape description techniques which are more bottom-up and constructive than the model-fitting approaches. The shape models used are very local and are detected by local matching. Larger-scale shape models, such as circles, rectangles, and complex irregular shapes are *built up* out of these locally detected pieces, using connectedness of boundaries or axes. Thus, local symmetry representations can identify a larger class of shapes more robustly.

Local symmetry representations based on symmetries of connected sections of boundary should be contrasted with more global symmetries based on matching sections of an image against itself. For example, in order to detect textures or motion, one might match an image, or some symbolic representation of an image, against itself with some translation, rotation, or possibly reflection. Such global matching symmetries within an image may also be useful in describing an image. However, they represent different types of properties from the region-forming local symmetries described in this thesis.

5.2. Building complete shape descriptions

Most shapes of common objects are too complex to be represented by a single local symmetry region. A local symmetry description of a pear, an airplane, a spanner wrench, a square, or a triangle involves several local symmetry regions joined together or cut out from one another. In order to be useful for recognizing and reasoning about shapes, the raw local symmetry analysis must be transformed into an analysis of the shapes as a set of subshapes that:

- Exhaustively cover the shape;
- Do not account for the same part of the shape in more than one way;

- Have explicitly specified spatial relationships to each other.

Bagley (1985) and Connell (1985) have recently done work on constructing such analyses for Smoothed Local Symmetry representations of shapes. Similar work needs to be done for analysis of shapes containing both Smoothed Local Symmetry regions and Local Rotational Symmetry regions. Since this is future work, I will only briefly sketch the issues involved.

One set of issues involves representing the relationships between the sub-parts involved in the analysis. First, the coordinate systems of different subparts must be related. This involves specifying direction, distance, and orientation of one subpart with respect to another. When two parts are attached to, adjacent to, or cut out of one another, these facts should be made explicit. A local symmetry region may represent either part of an object or part of the background. Two regions which share the same boundary are typically a region of some object and a region of the background surrounding that object. A region which is cut out of an object region must be part of the background. When the boundary of one region is directly connected to the boundary of another region, this fact should also be made explicit, as in the pear and bar shapes in Figure 1. Two regions whose boundaries connect must be two parts of the same object or two sections of the background.

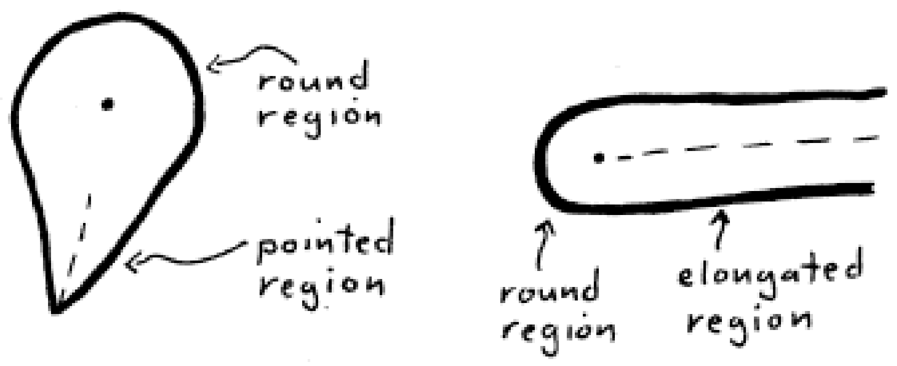

Figure 5-1. A pear shape and the rounded end of a rod. In both of these figures, the two symmetry regions smoothly extend each other's boundaries and region.

Between the primitive symmetry regions and shape models for natural classes (e.g. hammers), there are a number of simple and frequently occuring combinations of symmetry regions that should be recognized. For example, there are a number of standard ways of terminating an SLS region (cf. Brady and Asada 1984):

- open termination into some other type of region;
- blunt termination at the last rib;
- pointy SLS region continuing its boundaries;
- LRS region continuing its boundaries.

These four types of terminations are illustrated in Figure 2. Some common combinations of symmetries involve several competing local symmetry analyses. For example, a ring or a spiral has an analysis as a long SLS region, as well as an analysis as two LRS regions. A hexagon has several competing SLS analyses (the main axes plus corners), as well as an LRS analysis. The configuration of multiple competing analyses might be made explicit in the representation of these region combinations.

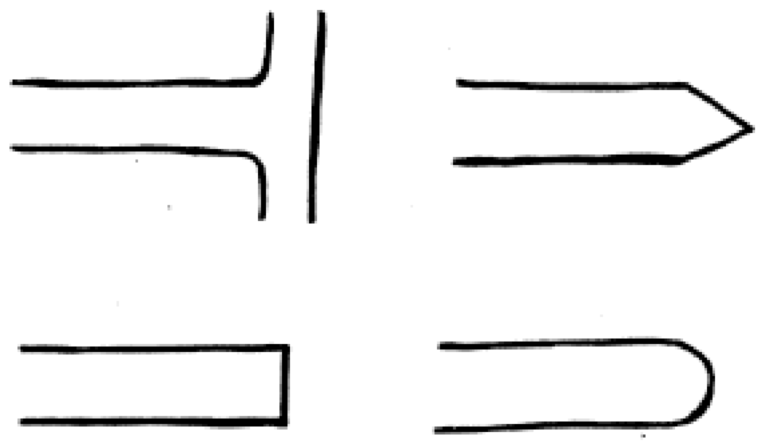

Figure 5-2. Four ways of terminating an elongated SLS region. Top: open termination and termination in a pointy SLS. Bottom: blunt termination and termination in a half-round LRS.

Finally, there seems to be a constraint that local symmetry regions involved in an analysis "not account for the same part of the shape twice". For example, sub-optimal LRS regions with centers near the perceived center of a region are not perceptually salient. Also, two SLS regions in the same analysis cannot in general overlap, as illustrated in Figure 3. However, it is not obvious how to make this constraint precise.

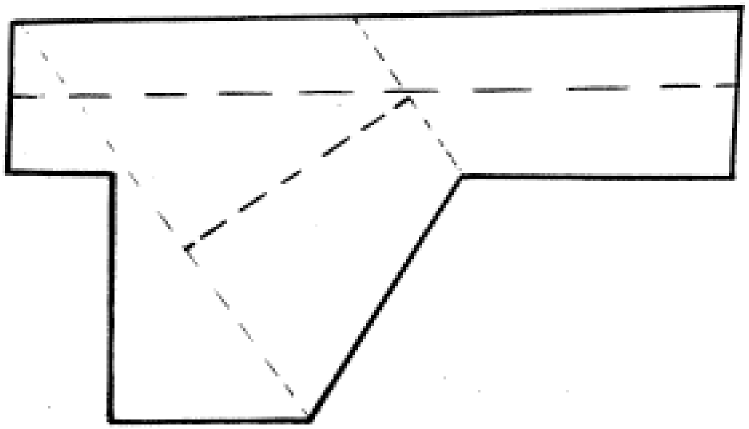

Figure 5-3. The axes of two possible symmetry regions are marked with dotted lines. These two regions are perceptually incompatible, presumably because they overlap.

It seems that a perceptually correct version of the constraint that symmetry regions not overlap involves consideration of:

- overlap in boundaries,
- overlap in 2-dimensional regions covered, and
- whether the two regions represent parts of the same object, or whether one is part of an object and one is part of the background around the object (either a region of empty space or a region of another object).

Note that whether a symmetry region represents part of an object or part of the background cannot be determined *a priori* from the input image. Rather, the organization of the image into a number of distinct objects and a background is one part of the task of creating a complete shape representation. Thus, constraints on region overlap should be viewed as consistency constraints on how a complete analysis of an image can be constructed.

Two, or even three regions can share the same piece of boundary. The possibilities are illustrated in There are two relationships that abutting regions can bear to one another and form part of a consistent description of the scene. The first possibility is that the two regions cover disjoint 2-dimensional regions, as illustrated in Figure 4. Note that the "two regions" considered here could in fact be parts of the same region, as shown by the spiral in Figure 5. The second possibility is that the two regions cover overlapping 2-dimensional regions and one region is cut out of the other as shown in Figure 6. Two regions are in conflict when they cover overlapping 2-dimensional regions and one is not a cut-out of the other, as Figure 3 illustrated.

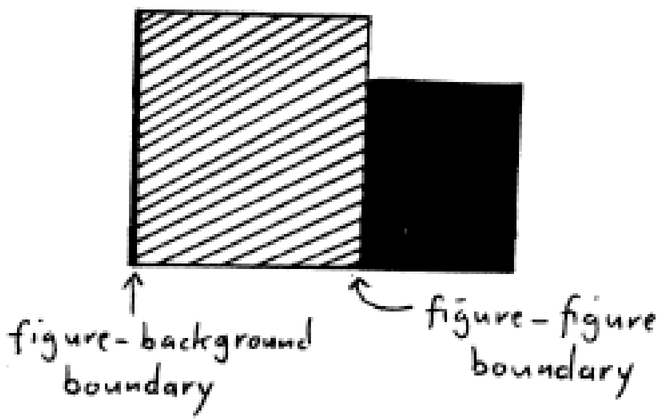

Figure 5-4. A boundary can be shared by two figures, or by a figure and the background.

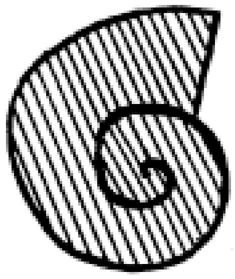

Figure 5-5. Self-adjacent spiral. This figure shares a section of boundary with itself.

This statement of the constraint, however, does not seem to be quite correct. The problem is that some types of regions seem to be able to co-exist even though they share boundaries and cover overlapping regions. For example, the corners of a rectangle seem to be salient, despite being in conflict with the main axis of the rectangle (cf. Brady and Asada 1984); the regions describing the pointed jaws of a spanner wrench seem to co-exist with the regions describing the round ends and square cut-outs of the wrench; an oval that is on the borderline between round and elongated can be described as one round region or as having two round ends. It is not clear that main region of such an ellipse and its more coherent round ends are peceptually in conflict. These examples are shown in Figure 7.

My LRS implementation uses the heuristic that two regions overlap incompatibly if they share half of their contour. As some of the output examples in

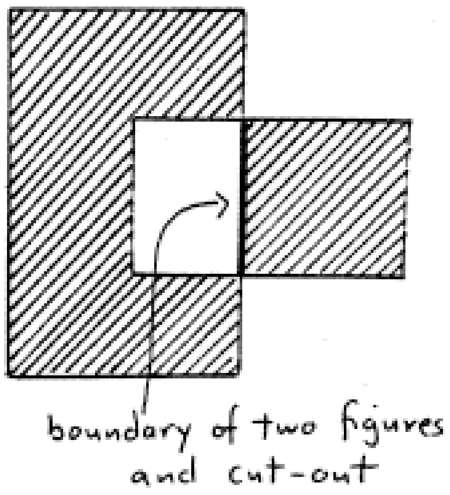

Figure 5-6. A region can share a section of boundary with a cut-out region. Where there are cut-outs, there can be more than two compatible regions sharing the same section of boundary.

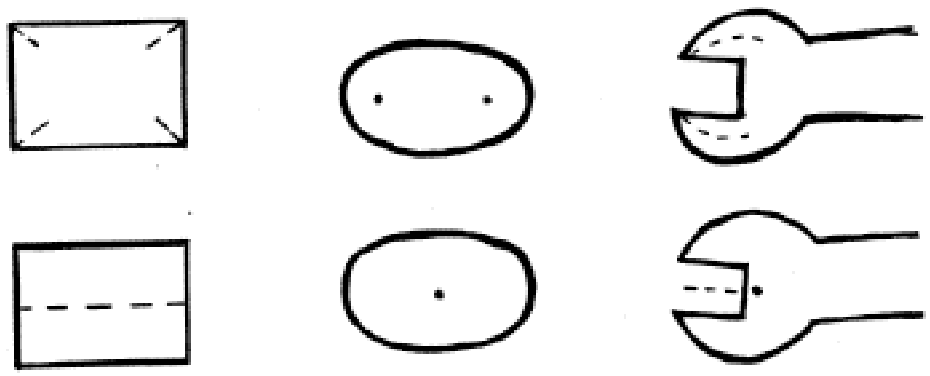

Figure 5-7. Regions can co-exist even though they overlap and share boundaries. The jaws of the spanner wrench co-exist with the round end and square cut-out. The ends of a fat oval co-exist with a description of the whole oval as basically round. The corners of a rectangle co-exist with its main axis.

Section 4.7 illustrate, this heuristic does not robustly detect when two regions are perceptually "basically the same" or when they overlap "too much."

5.3. Multiple descriptions

Marr and Nishihara (1978) state that the representation should be designed so that each shape has one canonical representation, so that processing using these representations need not search through multiple possibilities in matching shapes. The idea of limiting representations of a given shape to a small number is clearly reasonable. However, if shapes intermediate between two types of shape descriptions, e.g. round shapes and elongated shapes, are required to have a unique analysis, the shape representation will have a sharp change in representation at some point. In many such cases, the intermediate shapes are perceived by humans as varying smoothly from one type of shape to the other. The sharp discontinuity in the shape representation does not match human perceptions and the location of the change from one type of representation to the other must be chosen arbitrarily.

The solution in these cases is to allow intermediate shapes to have more than one representation. Suppose that the relative salience of the competing representations is allowed to vary continuously, so that the salience of one type of representation diminishes gradually as one moves towards shapes more representative of the other type of representation. So, for example, an SLS representation is very salient for rectangles, less so for ellipses, and not salient at all for circles. When one of the competing representations reaches sufficiently low salience, it should no longer be considered plausible at all. However, the change from an extremely low salience alternative to no alternative is not a sharp change in representation. Small changes to this salience threshold should make little difference to users of the representation. This controlled use of multiple representations for the same shape allows the representational system to accurately model human perceptions of similarity and smooth change (i.e. the representation is stable) and avoids use of arbitrary thresholds.

There are several situations in which there are smooth transitions between different types of shape models. First, there are two different types of local symmetry representations: Smoothed Local Symmetries for elongated shapes and Local Rotational Symmetries for round shapes. Secondly, there may be qualitatively different ways to analyze a complex shape in terms of sub-parts. Finally, larger-scale shape models such as the natural classes "cup," "vase," and "bowl" may grade smoothly into one another.

The Local Rotational Symmetry representation for round regions was specifically designed to overlap somewhat with the Smoothed Local Symmetry representation for elongated regions. For example, the oval shapes shown in Figure 8 vary smoothly in analysis depending on the proportions of the figure. The longer regions are analyzed as an elongated region with two round ends. The fatter regions are analyzed as one irregular round region. Intermediate shapes have both analyses, with varying degrees of relative salience. Similarly, Figure 9 shows the transition from a circle (clearly round) to a lozange shape (clearly elongated), via a hexagon which has both types of analyses. Similar examples can be constructed using only one shape model, e.g. Smoothed Local Symmetry regions, but matching it to the shape in different ways. Figure 10 shows gradual transitions from a rectangle to a square to a pointy diamond.

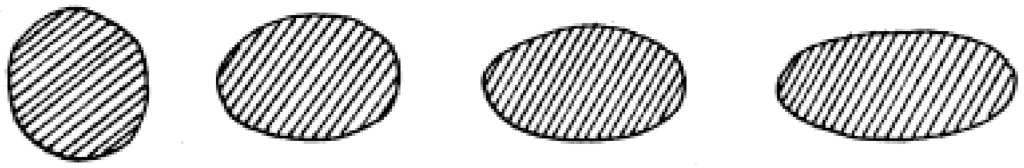
Figure 5-8. A circle, which is perceived as one round region, can be smoothly deformed into a long oval, which is perceived as a long region with two round ends. Intermediate figures may be perceived as ambiguous between the two types of descriptions.

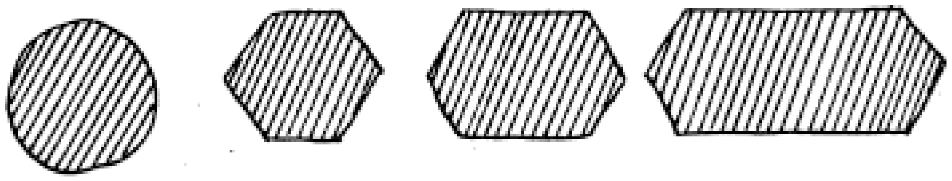
Figure 5-9. A circle can be smoothly deformed into a lozange shape. Intemediate shapes, such as the hexagon, can be described either as a round region or as an elongated region

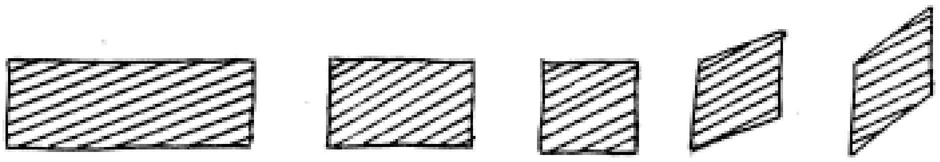
Figure 5-10. A rectangle can be smoothly deformed into a diamond. The rectangle has one most salient axis, the diamond has a different type of most salient axis. The square, which is intermediate between the two shapes, has several salient axes.

Multiple competing representations also occur in analyzing complex shapes in terms of simpler ones. For example, Bagley (1985) describes a system for building representations of complex polygonal shapes in terms of a main shape and shapes attached to it or cut out from it. He shows examples of sets of complex shapes that vary smoothly from one prefered analysis to another, with intermediate shapes having both analyses as options. Figure 11 shows one of his examples. (See also Hollerbach 1975.)

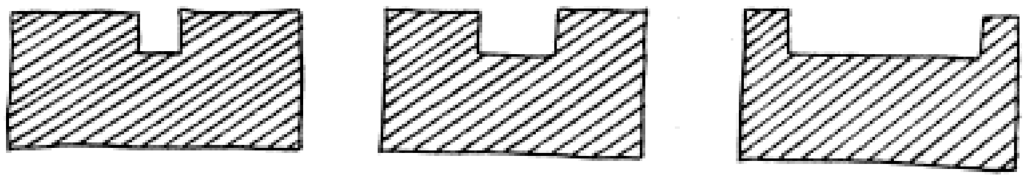

Figure 5-11. Complex shapes may have multiple interpretations. The figure on the left is most naturally interpreted as a rectangle with a piece cut out of it. The figure on the right is interpreted as a rectangle with two tabs attached to it. The middle figure is intermediate.

In constructing a complete analysis of a shape, there may be a tradeoff between an analysis in terms of a larger number of local symmetry regions with simple patterns of parameter variation and an analysis in terms of a smaller number of regions with complex patterns of parameter variation. For example, Figure 12 shows an SLS region that could be described as either one region with complex variations in width or as a number of regions concatenated. The first description would be most useful if this were an arbitrary complex shape produced on a lathe, e.g. a decorative table leg or door post. The second description would seem more appropriate if the individual pieces had distinct functional roles, e.g. if this were a complex tool.

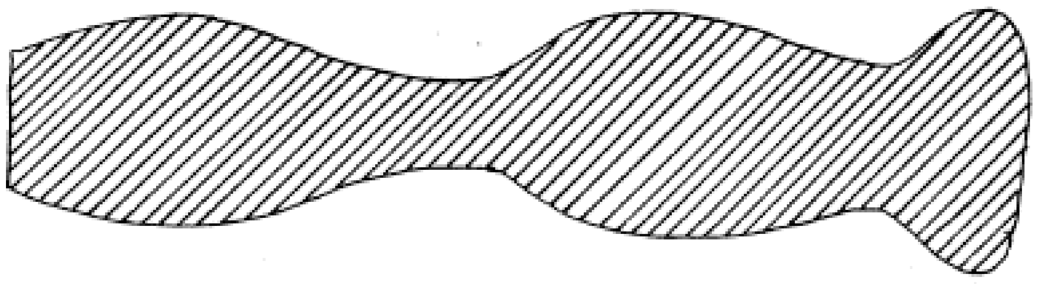

Figure 5-12. A complicated SLS region. Such a figure could be interpreted as one region with complicated changes in width, or as three regions with simpler descriptions.

A similar effect seems to hold in the way people use natural language words to refer to objects. The data presented in Labov (1973) suggest that there is a smooth decay in how well a shape is perceived as member of the class named by some English word, such as "cup" or "vase," as the shape is slowly made less and less like a good exemplar of that class. In labelling shapes with natural language words, people can use a particular word to refer to an object, or they can decide that there isn't any term that really covers the shape. For instance, the shape in Figure 13 doesn't really look like anything in particular. Adding an option of "matches nothing" does not, however, change the argument about sharp transitions. The transition from a shape that has a good label to a shape that does not is still smooth.

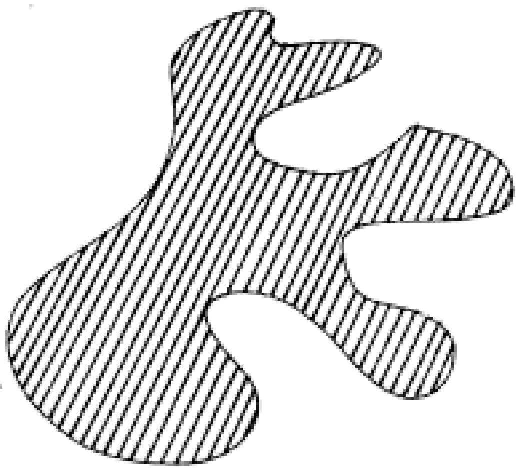

Figure 5-13. A shape that doesn't look like anything in particular.

5.4. Region color

The Smoothed Local Symmetry code described in Brady (1983), Brady and Asada (1984), and Heide (1984) incorporates a constraint which I will call the *Region Color Constraint.* The idea behind this constraint is that a local symmetry between two sections of boundary is only salient if the color of the region on the interior sides of the boundaries (i.e. the sides toward the symmetry axis) is similar. For example, the symmetries shown in Figure 14 (left) are salient symmetries, but the symmetry shown in Figure 14 (right) is not. For a round region, a similar constraint seems to hold, as shown in Figure 15. Figure 16 illustrates the fact that

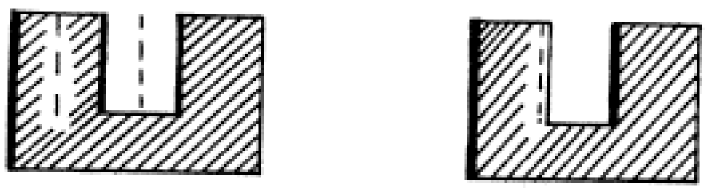

Figure 5-14. The left-hand figure shows a symmetry down one elongated region of a shape and also a symmetry in an inlet of the shape. These axes are perceptually salient, because they involve symmetries whose two sides match in color. The right-hand figure shows a symmetry between the outside boundary of the shape and "the wrong" side of the inlet. This symmetry is not salient, because the two sides of the symmetry do not match in color.

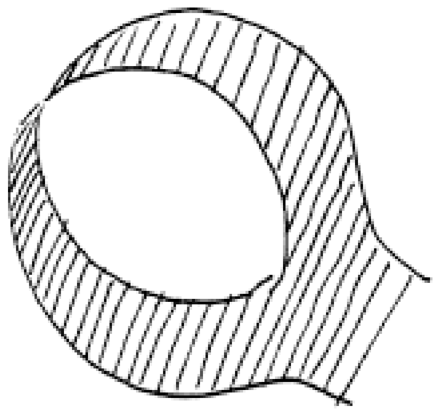

Figure 5-15. There are two plausible LRS regions in this figure: the inside of the jaws and the outside of the jaws. Possible symmetries involving the inside of one jaw and the outside of the other are not perceptually salient, because the sides of the symmetry do not match in color.

the constraint is only on the color of the interior of the local symmetry region, not on the exterior.

The idea behind this constraint is that region color can be used as a heuristic for determining what regions in an image are part of the same object and what regions are part of the background around that object. Most of the images used in testing the Smoothed Local Symmetry representation contain one or more dark objects on a light background or one or more light objects on a dark background. Therefore, in these images, region color is a good indicator of whether a region is figure or background. Furthermore, there are only two colors to worry about in any particular image: light and dark. Then, since a local symmetry region defines a 2-dimensional shape or part of a shape, the interior of the symmetry region should be all the same color or perhaps smoothly varying in color. The

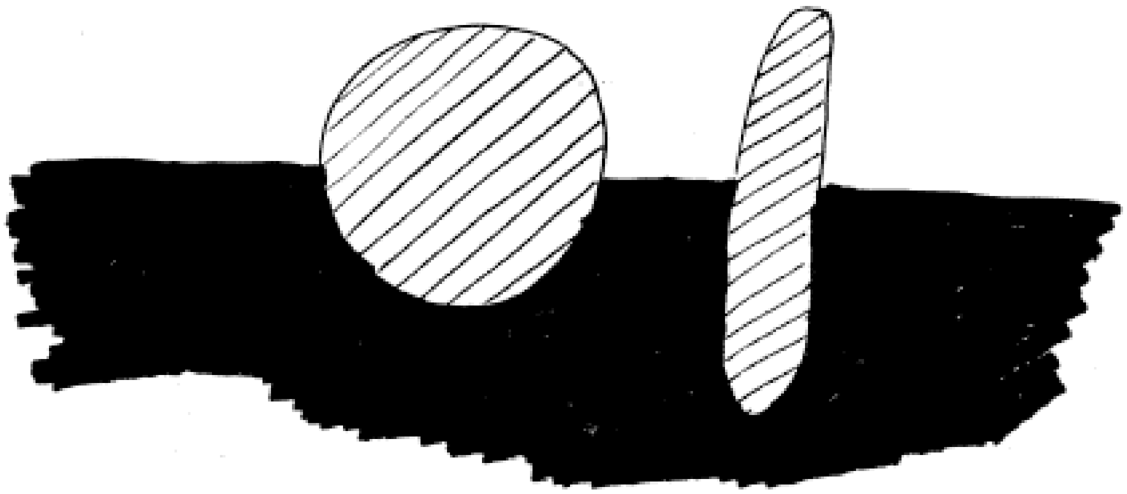

Figure 5-16. A monochrome grey ball and a grey rod, each half buried in a black region. The constraint on region color must apply only to the *interior* sides of the boundaries of the symmetry regions.

problems appear when one tries to extend this idea to grey-scale images containing regions of more than two distinct levels of intensity.

In the implementations of Smoothed Local Symmetries (Brady and Asada 1984, Heide 1983) the normals at boundary points are directed, e.g. always pointing towards the darker side of the boundary. The region color constraint was implemented by requiring that the normals at the two boundary points in a local symmetry either both point outwards or both point inwards. This heuristic will not in general produce the right answers. For example, in Figure 17, the grey stripe seems like a reasonable SLS region, although the normals on one side point inwards and the normals on the other side point outwards. Thus, a perceptually correct implementation of the constraint would have to refer directly to the color of the region near the boundaries, rather than using the directions of the normals. One unsolved problem in implementing a correct version of the constraint is determining what the exact conditions are for the interior sides of the two sections of boundary to be "close enough to the same color".

In aggregating local symmetry pairs into regions, a change in interior color between two adjacent pairs, i.e. an internal color boundary in the region, seems to be a reason to consider dividing the region in two. However, in such a situation, it is also possible to ignore the color boundary and make one symmetry region. For example, the stripe in Figure 18 can be construed as either one region

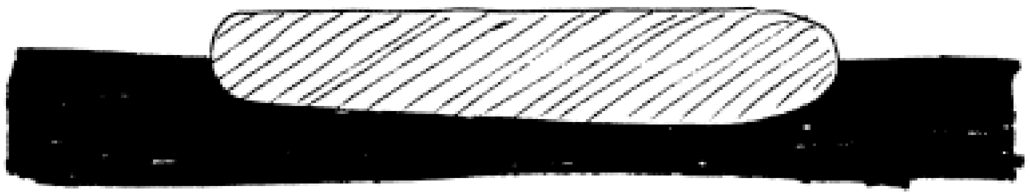

Figure 5-17. A grey stripe with a white background to one side and a black background to the other. This example shows that comparing the direction of color change on the two sides of a symmetry region is not a perceptually correct statement the constraint on region color. In this figure, one side of the symmetry is darker on the inside and the other is darker on the outside. Nevertheless, the symmetry region is perceptually salient, because the color on the interior sides of the boundaries is the same.

(by ignoring the color change) or as two regions, divided at the color boundary. Round regions can also be divided up by internal color boundaries, with the same optionality, as in Figure 19. The existence of such figures makes it unclear how to formulate the constraint for round regions, because for these regions we cannot neatly separate the question into constraints on the points in the same symmetry pair and constraints on adjacent pairs. Worse, even the constraint on the two points in an SLS symmetry pair does not hold strictly. For example, the symmetry along the stripe in Figure 20 can be continued along the boundary of the black tab, despite the violation of the color constraint.

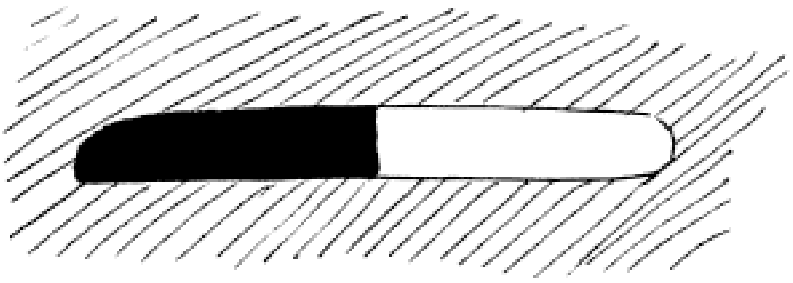

Figure 5-18. A stripe, black for one half and white for the other, on a grey background. This stripe can be peceived as either one region or two.

Because of these problems with the definition of the region color constraint, I have not implemented any form of it for Local Rotational Symmetries. Obviously, more research needs to be done into the exact form of the perceptual constraint on region color.

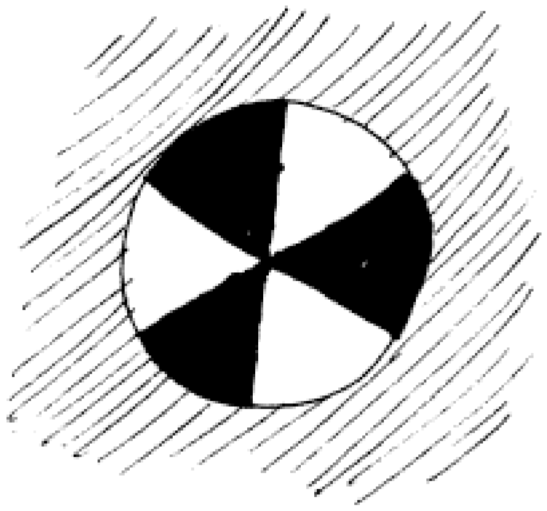

Figure 5-19. A beachball figure. Round regions divided into sections by color boundaries can still be interpreted as one region, despite the difference in color.

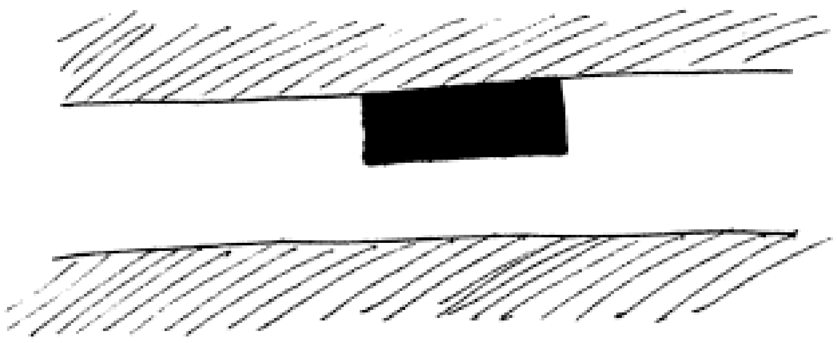

Figure 5-20. A white stripe with a black patch on it. The whole stripe, including the patch, can be interpreted as one elongated region, despite the change in color along part of the boundary.

Chapter 6: Multi-scale representations

The shape representations discussed in Chapters 3 and 5 all represent shapes with a fixed, finite amount of detail or resolution. For recognizing and reasoning about objects and scenes, we need to be able to represent a shape with any arbitrary amount of resolution, in order to be able to distinguish small differences between objects or scenes. It is also necessary to be able to abstract away from detail when it is not needed, representing only the most important aspects of an object or a scene. These two requirements are best met by having a series of representations of an object with varying amounts of detail. If the representations at different levels are related, so that information computed at one level can be transfered to other levels, a multi-scale representation allows one to achieve several apparently incompatible goals at the same time:

- Results are highly accurate and representations are highly detailed;
- Computations and representations are stable under small changes in input;
- Computations have data dependencies that are non-local in the original input and so can detect large features;
- Computations have only local data dependencies in the multi-scale data structure used for computation and thus are efficient, particularly on parallel hardware.

Multiple scale representations have been proposed, to achieve varying combinations of these goals, by a wide variety of researchers. Pyramid-shaped processing structures have been widely used in image processing. For a summary of this literature, see Tanimoto (1978). The main goal in this work is to use pyramid-shape arrays of processing elements to do efficient calculation. On the vision end of things, Terzopoulos (1984) uses multi-grid relaxation to produce efficient algorithms for surface reconstruction. Again, although he is interested in producing output at multiple scales of resolution, much of the focus is on using the pyramid structure to allow efficient computation.

There has also been considerable work done on finding and interpreting edges in images at multiple scales, surveyed by Torre and Poggio (1984). These researchers find representing edges at multiple scales interesting because the set of edges in an image at all scales may be a complete representation of the image and also because multiple-scale representations allow both good detection of edges (at coarse scales) and good localization of edges (at fine scales). For example, the edge finder described in Canny (1983) finds edges in an image at multiple scales and matches these edges across scales to yield a unified map of edges at all scales, using an explicit mathematical theory of how an edge at one scale will be reflected at the next coarser scale. Witkin (1983) proposed summarizing multiple-scale representations of a signal into what he calls a "scale-space" representation. In this representation, the shape of the signal across scales is summarized qualitatively by describing: the set of features in the signal, the range of scales at which each feature occurs, and the scales at which features split or merge. In his work, the features considered are inflection points of a one-dimensional signal.

There has been less work done on multiple-scale representations using features that are higher-level than edges. Asada and Brady (1984) developed a system for locating sharp changes in orientation along boundaries (corners, for example) and tracking these features across scales to produce a representation analogous to Witkin's representation for one-dimensional signals. Ponce and Brady (1985) extend this work to representing the shape of 3-dimensional surfaces. Hoffman (1983) tried to locate "natural scales" in multi-scale representations of boundary curves. Crowley (1982) locates and tracks features for describing regions across scales, but he uses relatively *ad hoc* methods for relating representations at different scales.

On the other side of the fence, researchers working on planning and reasoning have been talking about representing situations or objects at multiple levels of resolution or abstraction, although there are only a few systems that actually use this idea. Planning can be made more efficient if it is done first with a coarse-scale representation of the problem and then fleshed out in more detail, most recently proposed by Allen and Koomen (1983). They also propose using a hierarchical organization of time intervals in order to restrict explicitly stored temporal relations to intervals local to one another. Such a locality restriction would allow algorithms for reasoning about temporal relationships to run efficiently. There has been some limited work in using representations which involve *qualitative* changes in representation between finer and coarser scales, for example Weld (1984) and Patil (1981). Taylor (1977) and Dowty (1979:163-173) have discussed problems in the that have been discussed in natural language semantics that could be resolved by using multi-scale representations.

Shape representation lies somewhere in between the low-level visual processing and high-level representations for reasoning and natural language understanding. Ultimately, shape representation systems must serve as the connection between these two types of processing. Very little work has been done on creating shape representations at multiple levels of resolution. Marr and Nishihara (1978) propose using multi-scale representations for objects, but their proposal is not very detailled. Brady and Asada (1984) build Smoothed Local Symmetry Representations for shapes at multiple scales of resolution, but they do not relate the representations at adjacent scales. In building an efficient implementation for computing Local Rotational Symmetries, I have had to reconsider the way in which Brady and Asada computed multiple scale representations. The goal in designing the multi-scale LRS representation has been not only to use multiple-scale representations for efficiency and compatibility with theories of low-level vision, but also to be sensitive to the needs of reasoning and natural language understanding systems that might be using its output. My implementation of Local Rotational Symmetries uses a local multi-scale computation with communication between scales. Detection of cross-scale features and patterns has not yet been implemented, although the inter-scale matching provides support for adding such analysis.

This chapter will discuss issues in computing and using multi-scale representations in shape representation. These issues include:

- Types of abstraction from detail;
- Inter-scale matching, communication between scales, and cross-scale features;
- Locality of computation;
- How attention may affect multi-scale processing.

In discussing each of these issues, I will point out what choices have been taken in implementing a system for computing Local Rotational Symmetries at multiple scales.

6.1. Types of scale abstraction

Coarser-scale representations are ideally supposed to contain only the "important" or "stable" or "overall" features of finer-scale representations, abstracting away from detail. In practical reasoning, the importance of some feature is determined by a large number of factors, including functional importance as well as size. For example, the difference in the shape of the business ends of a Philips and a regular (slotted) screwdriver is crucial in explaining the functional properties of these tools and would appear even in relatively coarse-scale representations of screwdrivers, despite the small size of the features (Connell and Brady 1985). Even when only size is used to determine importance, the cut-off for how small a feature is represented may or may not be a constant across the entire representation. For example, at the instant that one is looking at a chair, the chair may be represented with high resolution while the context around it is only more coarsely blocked out.

In the early analysis of a visual image, the only index of importance available is size. Therefore, the techniques for abstraction discussed here will only use size as an index of importance. However, nothing prohibits later reasoning from readjusting the results of these early analyses or influencing processing of shape as it is going on. While early visual processing (e.g. stereo) seems to be more or less independent of the content of the image, decisions about which objects to focus on in a complex scene (e.g. a large cluttered room) are clearly dependent on the viewer's goals. How far down the influence of higher-level goals extends in visual processing is an empirical question which I cannot answer here. The algorithms for computing multi-scale circular region symmetries do not depend on higher-level information, but could make use of such information to guide processing, were it available. Similarly, these algorithms do not crucially depend on the resolution being constant over the entire field of view.

The abstracted or coarse-scale representations of an object that people use often involve qualitative differences in representation compared to finer-scale views of the same object. For example, the fact that brushes can clean particles off of surfaces depends on the outside of a brush being made up of a large number of bristles. Thus, a fine-scale view of bristle texture can be used to explain why a floor brush is different from a blackboard eraser. However, understanding what shapes of brushes are useful for what types of cleaning tasks also requires considering an abstracted representation of the brush which shows just its overall shape, with the surfaces of the shape marked as to whether they are made up of the business ends of bristles, the attachment ends of bristles, or the sides of bristles. For example, a bottle brush can be represented as a cylindrical bristle section attached around a long thin handle, with the entire exterior of the cylinder being composed of the business ends of bristles. A floor brush, on the other hand, has a bristle section with all the business ends of the bristles on one flat side and the other ends of the bristles attached to a flat back. In fact, the coarse-scale representation of a floor brush is very similar to the coarse-scale representation of a blackboard eraser, a fact which makes it easy to explain why these brushes require similar hand motions and are used to clean similar shapes of surfaces whereas a bottle brush requires rather different hand motions and is used on different types of surfaces. Reasoning about the functions of these brushes directly from individual bristle locations without using abstracted bristle surfaces would be extremely slow and tedious.

A different type of example involves edge texture, as illustrated by the leaf with a serrated edge shown in Figure 1. Standard terminology for describing leaf shapes (cf. Petrides 1972) separates the description of a leaf into a description of its overall shape (oblong, narrow, heart-shaped, long-pointed) and a description of the texture of its edges (wavy, toothed, double-toothed). A multi-scale local symmetry analysis describes both the coarse-scale shape and the fine-scale edge texture, as shown in the oak leaf example (Figure 1-4) and the cog example (Figure 1-6) from Chapter 1. If analysis is done at only one scale of representation, it is not possible, in general, to pick out both the serrations and the overall axis of the leaf. If the serrations are visible and don't exactly line up on the two sides, the overall symmetry axis will be disrupted.

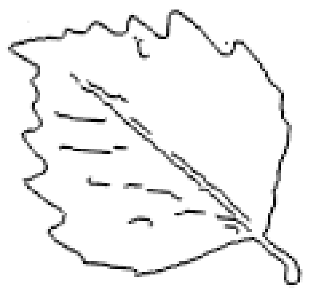

Figure 6-1. A serrated leaf. The standard way to describe such leaves is in terms of their overall shape plus the shape of the texture on their edges (e.g. serration).

In both the brush and the leaf examples, the coarser and finer scale representations were *qualitatively* different. For example, there may have been regions or other features in the coarse-scale representation that were not present in the fine-scale representation. Such qualitative changes are useful not only in representing shape, but also in more abstract domains. For example, recent work by Weld (1984) and Patil (1981) uses qualitatively different representations to reason about complex situations in biochemistry and medical diagnosis. The same ideas could also solve the problems in representing non-homogeneous actions discussed by Taylor (1977) and Dowty (1979). The specific examples that Taylor and Dowty discuss are in linguistic semantics, but the axioms used by Allen (1984) for representing actions for practical reasoning have the same problems.

There are several techniques for abstracting away from small detail in a visual image:

- Feature dropping;
- Boundary smoothing;
- Image smoothing;
- Threshold changing.

Feature dropping techniques take a symbolic representation of an image and remove regions or other features which are small or unimportant according to some criterion. This abstraction technique is extremely important in higher-level learning and reasoning. However, by itself, feature dropping cannot account for qualitative differences in representation between finer- and coarser-scale views of an object.

Threshold changing involves changing the setting of a threshold that is used to select which features are "good enough" or "salient". For example, region boundaries in images differ in their strengths, i.e. in how much the intensity changes from one side of the boundary to the other. Relaxing strength thresholds on boundaries results in more detailed analysis of the regions in an image. Blake (1983a, 1983b) uses such a threshold changing technique to produce a series of representations of the boundaries of an image. Like feature dropping, threshold changing is a useful technique for varying the amount of detail in a representation. However, it cannot be the primary means of changing the scale of a representation. First, like feature dropping, it can only remove features as one tightens the threshold, not add them. Furthermore, while the sharpness of a boundary is one indicator of its importance within a representation at a particular scale, the size of regions, rather than their sharpness, seems to determine the differences between representations at different scales. Finally, representations based on threshold changing are sensitive to blurring an image or introducing noise.

The other techniques for creating representations at multiple scales of resolution involve smoothing the image. For a one-dimensional signal, such as that used in Witkin (1983), there is only one alternative: smooth the signal and then extract significant features at each scale. However, for two-dimensional images, there are at least two alternatives:

- Smooth the image and extract boundaries at each scale (used by Crowley 1982 and Canny 1983).
- Extract boundaries from the image and smooth these boundaries (used by Asada and Brady 1984, Brady and Asada 1984, and Hoffman 1983).

The advantage to extracting boundaries first and then smoothing them is that it allows region boundaries from grey-scale images and boundaries shown in line drawings to be analyzed in exactly the same way. However, current off-the-shelf boundary smoothing techniques require that boundaries be simple connected curves. If there are gaps in the boundaries, smoothing will not operate across the gap. Further, it is not obvious how smoothing should proceed across points at which several boundaries meet, e.g. when there is an internal color boundary. Both of these types of imperfections occur frequently in the output of our edge finder and also in line drawings. In the absence of a robust algorithm for extracting "the correct" connected boundary of a figure, these defects cause serious problems for the boundary smoothing approach.

Further, the smoothing done by existing techniques for contour smoothing does not match intuitive judgements about the overall shape of objects. For example, consider the image of a drain strainer shown in Figure 2, with the boundaries extracted by the edge finder. The internal boundaries of the holes in the strainer seem to be detail relative to the exterior boundary of the object, but smoothing the boundaries in this image will not eliminate the internal boundaries. Smoothing the grey-scale image, however, extracts just the overall shape of the object, as shown in the same figure. In general, smoothing the grey-scale image produces effects which seem to correspond well to intuitive judgements of the overall shape of objects.

Smoothing the grey-scale image also has the advantage that it has the same effect as the blurring caused by seeing the same scene from a greater distance. Thus, a multi-scale representation created by smoothing the image will be invariant across changes in size. This type of smoothing is also the technique used by

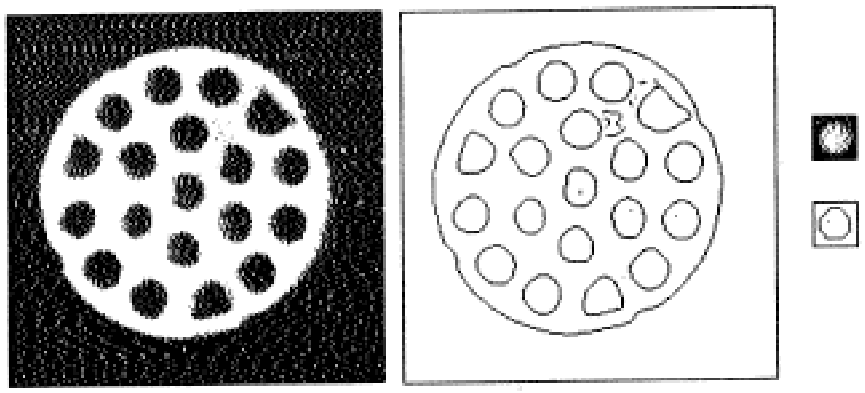

Figure 6-2. The grey-scale image of a metal drain strainer at a fine scale of resolution and at a coarser scale of resolution. The boundaries for both scales are also shown. Smoothing the grey-scale image abstracts away from the detail of the textured interior of the figure. Existing techniques for smoothing boundaries cannot remove the boundaries of the holes.

low-level vision algorithms, such as the surface interpolation algorithms described by Terzopoulos (1984). Therefore, the technique I use in computing representations of the shape of an object or a scene at multiple scales is to first smooth the grey-scale image and then extract region boundaries at each scale. Smoothing is done with a finite mask approximating a Gaussian filter. The smoothed image is sampled at a rate proportional to the size of the Gaussian to create the coarser-scale image. I use this technique in my implementation of Local Rotational Symmetries. The Smoothed Local Symmetry examples shown in this thesis use a version of the Brady and Asada (1984) implementation that has been altered to use grey-scale smoothing. This technique was used to produce qualitatively different coarse- and fine-scale representations of images such as the cog and the oak leaf shown in Figures 1-4 and 1-6.

One disadvantage to smoothing the grey-scale image is that it performs poorly when the image contains extremely thin lines, because they tend to disappear before the shape they enclose is smoothed. Further, when there are boundaries with only a small but sharp change in intensity, the boundaries may not be picked up in smoothed versions of the image. In either of these cases, images may have large regions whose boundaries will not be detected in smoothed form. However, I submit that since smoothing grey-scale images seems to be perceptually correct for grey-scale regions and existing boundary smoothing techniques are not, the right solution is to find a way to emulate the effects of this type of smoothing on isolated boundaries, such as the boundaries in line drawings. The general idea of such an approach would be to assume that the boundary is separating two contrasting regions, where the contrast in this case must be introduced artificially, and figure out what the effects of smoothing would be. In this context, I should note that in some cases, e.g. a tangle of twine in a cluttered room, thin regions *should* disappear as coarser-scale representations are created. Whether thin regions are retained in coarser-scale representations may depend on factors other than their size and proportions.

A second problem with smoothing the grey-scale image is that when two objects are near one another, the smoothing will blur one into the other. In some cases this is desirable, for example when one is trying to describe larger-scale patterns of arrangement of objects or coarser-scale shapes. In the initial coarse-to-fine analysis of a situation, I do not know of any way to distinguish "two figures" from "one figure" so that the two cases could be smoothed differently. However, *after two figures have been identified*, it should be possible to re-do the smoothing of each figure independently, without interference from the other figure. Since Gaussian smoothing (with a finite filter) has only local data dependencies, this would not require extensive re-computation. As in the case of extremely thin regions, it seems as though whether two objects should be separated or blurred into one another depends on details of the emerging shape analysis and perhaps on the goals of the reasoner using and directing the shape analysis.

6.2. Relating scales

It is not sufficient to analyze an image independently at a series of scales. First, when an object has an overall shape and more detailed features, people seem to be able to relate their locations. For example, the axes of the lobes of the oak leaf discussed in Chapter 1 (Figure 1-4) are related in location and orientation to the main axis of the leaf. Secondly, efficient computation of detailed representations for large regions or other features requires passing information between coarser and finer scale. Thirdly, if representations at different scales are not related, the representation is sensitive to the details of which discrete set of scales of resolution were chosen, as noted by Witkin (1983). Finally, a qualitative description of the stable representations of parts of an image and how they are related in scale is more informative than a simple listing of representations at different scales (also noted by Witkin).

The first task that must be done in order to relate representations at different scales is to match corresponding features at adjacent scales. There are two different reasons for matching features at adjacent scales:

- Detecting cross-scale properties of representations, e.g. which features are stable across a range of scales;
- Using coarse-scale representations to guide attention in processing an image at fine scales.

Passing information between different scales is a relatively well-known technique in low-level iterative algorithms. For example, the algorithm for surface interpolation described by Terzopoulos (1984) passes information from coarser to finer scales. Information from coarser scales can be used either to add more global information to finer-scale analysis, or to focus finer-scale processing on just regions that seem interesting at a coarser scale. Systems for tracking higher-level features across scales (Asada and Brady 1984, Witkin 1983, Canny 1983, Crowley 1982) have not used coarse scale analyses to guide finer-scale analysis, but generate the analyses at each scale independently and then combine them.

A program that tracks features across scales needs two theories of how features change between scales:

- a theory of how a feature changes smoothly between scales;
- a theory of abrupt changes in features between scales.

The theory of smooth changes specifies when a feature at one scale should be considered a manifestation of "the same" cross-scale feature as some feature at an adjacent scale. For each cross-scale feature, the program specifies the range of scales at which the feature occurs and the scales at which the feature disintegrates or disappears. The length of the range of scales at which a given feature is detected can be used as an index of how stable or salient that feature is in the overall analysis, as Witkin (1983) does. It may also be possible to determine the scale at which a particular feature shows up most strongly or coherently, as Crowley (1982) and Hoffman (1983) try to do. The theory of abrupt changes specifies the ways in which features can merge, split, disappear, or otherwise abruptly change between scales. When a feature merges with another feature, splits into two features, or bears some other relationship to features that replace it when it disappears, these relationships are explicitly noted in the multi-scale represention.

Theories of smooth change in features are generally more or less straightforward. For simple features, an exact mathematical analysis of possible changes can be done. For example, Canny (1983) does a mathematical calculation of the predicted shape of a fine-scale edge at a coarser-scale in order to determine whether it should be matched with a given coarse-scale edge. The current LRS implementation uses a theory of smooth correspondence between LRS regions to pass suggestions from one scale of analysis to the next finer scale. Since this implementation only uses the locations of boundaries and not their strengths and since it works from coarser scales to finer scales, Canny's results for boundary matching could not be used directly. In order to pass information from one scale to another, I use empirically determined estimates of how much displacement to expect between a boundary or a center location at one scale and a corresponding boundary or center location at the next finer scale. The correspondences used in doing the suggestion passing could also be used to match regions found at one scale with regions found at the next finer scale, in order to produce a type of scale-space analysis of the LRS regions. This has not yet been implemented, although its feasibility is obvious from the smooth changes in analysis between different scales of the LRS analysis of test images (see Section 4.7) and from the fact that the suggestion passing mechanism works.

Abrupt changes include features disappearing at coarser or finer scales (Witkin (1983) explicitly rules out disappearances), two fine-scale features merging at a coarse scale, and other types of changes. (In general, two coarse-scale features do not merge at a finer scale.) Neat mathematical theories of either of these phenomena are easiest to build for low-level features, such as the inflections that Witkin tracks. The set of primitives that Asada and Brady (1984) and Ponce and Brady (1985) use for describing boundaries is richer than Witkin's primitives. Therefore, in their representations, the possible relationships between a coarse-scale primitive and one or more finer-scale primitives are more complex. For higher-level features, such as LRS regions, the possible types of abrupt changes between scales are yet more varied. When a coarse-scale region disintegrates at a finer scale, the regions that will be generated to describe corresponding sections of boundary in the fine-scale representation are related to patterns of parameter change in the coarse-scale region. For example, a round region with a smooth periodic variation in its radius will break apart into a series of small round regions. However, it is not clear to me whether there is any reason to work out all possible relationships in detail: it suffices to explicitly observe the abrupt changes when doing an analysis of an image.

I should note that, in the current LRS implementation, when the analysis changes qualitatively between scales, there are often several intermediate scales at which the fine-scale and the coarse-scale analyses co-exist, in somewhat degraded form. This behavior resembles the behavior of competing shape models on intermediate shapes: the two analyses co-exist for several scales and each analysis degrades smoothly as the scale is varied, resulting in a combined analysis that is stable under change in scale. This behavior seems to be different from the behavior of simple primitives, such as the inflections tracked by Witkin or the curvature primal sketch primitives of Brady and Asada. These simpler primitives split, merge, or disappear abruptly rather than gradually fading off.

When there are qualitative changes in representation between different scales of analysis, the high-level symbolic representations at each scale should be related. The relative spatial locations and orientations of axes and centers of regions at adjacent scales should be specified, possibly in exactly the same way as relative positions of regions at the same scale. A good example of this is a lobed leaf, such as the oak leaf shown in Figure 1-4. Because the lobes are not exactly symmetrical, the main axis of the leaf can only be extracted at a coarse scale of representation. The axes of the individual lobes can only be extracted at somewhat finer scales. However, specifying the pattern of attachment of the lobes to the main axis of the leaf is important in describing different types of lobed leaves (cf. Petrides 1972). Therefore, the joins between these primitives should be related, despite the difference in scale.

6.3. Locality of computation

It is not practical to compute relationships such as Smoothed Local Symmetries between all pairs of boundary points or other items in an image. Such global exhaustive pairwise computation for a 2-dimensional image will grow as $O(n^2)$ in the area of the image, even if the computation for any pair of features is constant. Similarly, computation of Local Rotational Symmetries for all boundary points and all possible centers in an image grows as $O(n^2)$ in the area of the image. This is true even if the center locations considered are restricted to locations close enough to the set of boundary points that a boundary made from these points could span some minimal percentage of the angles around the center. Data compression techniques such as the technique of approximating the contour by a set of line segments and circular arcs that Brady and Asada (1984) use can produce a substantial linear speedup in the computation for a fixed scale of resolution but do not change the rate of growth as the total number of boundaries in the image grows. Not only does computation time grow as $O(n^2)$ in the number of boundary segments, but a change in any part of an image or an addition of an additional section of image from another view requires recomputation of symmetries affecting all segments in the image.

The solution is to impose *locality of computation*: restrict exhaustive computation of Local Rotational Symmetries to boundaries and LRS center locations that are within some fixed distance of each other. Similarly, computation of Smoothed Local Symmetries should be restricted to pairs of boundary points local to one another. Not only is locality necessary for efficient computation of symmetry regions, but the symmetries eliminated by locality seem not to be perceptually salient. Since the symmetry computation runs at multiple scales of resolution, such a restriction does not prevent the algorithm from finding regions of arbitrarily large size: the boundaries of large regions will be local to one another at a sufficiently coarse scale. Furthermore, once a symmetry region has been hypothesized at a coarse scale, it can be efficiently tracked down to finer scales of representation. Thus, a local multi-scale computation in which information is passed between scales has the following properties:

- it is accurate;
- it is efficient; and
- it is able to detect large features.

The restriction imposed by locality is that, in order to be salient, a region must show up as salient at a scale coarse enough that the symmetries forming the region are local. That is, if the region is elongated, it must show up at a scale at which its width is small. If the region is round, it must show up at a scale at which its radius is small. My experience with symmetry representations of objects indicates that, at least to a first approximation, the potential symmetry regions pruned by the locality restriction are not perceptually salient. For example, Figure 3 shows the smoothed local symmetries of the oak leaf from Chapter 1 at a fine scale of resolution. The symmetries near the boundaries seem perceptually salient: they mark the axes of elongated or pointed regions in the figure. However, symmetries whose boundaries are far apart relative to the area of the symmetry region, such as many of the symmetries with axes near the center of the leaf, are not perceptually salient. The axes of these symmetries strike people as meaningless noise in the analysis. It is not just that people consider these regions counter-intuitive: people seem perplexed that one would even consider symmetries between two random pieces of boundary in totally unrelated parts of the image.

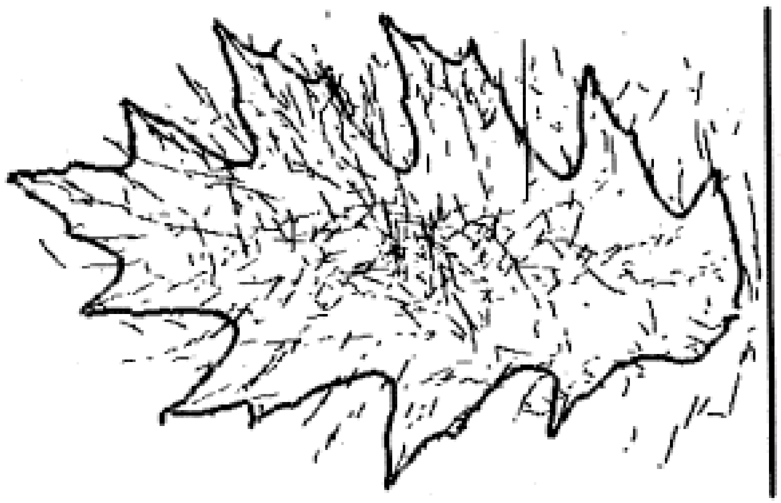

Figure 6-3. The Smoothed Local Symmetries of an oak leaf. The darker lines are the boundaries of the leaf and the finer lines are the axes of Smoothed Local Symmetries. Symmetries whose boundaries are close to one another compared to their lengths seem salient, whereas symmetries whose boundaries are far apart for their lengths seem like random junk in the analysis. This is perceptual evidence for locality of computation. (The vertical line represents a color boundary in the background on which the leaf was photographed.)

One description of why non-local Smoothed Local Symmetries are bad is that to be perceptually salient, an SLS region must have a low enough ratio of width to length (the so-called "aspect ratio" of the region). For round regions, there is an analogous measure: the angular length of the boundary around the center of the region. Heide (1984) and Connell (1985) use aspect ratio to filter out undesirable symmetry regions after they have been hypothesized. However, it is not clear how to use this measure to avoid computing most of these regions in the first place. For regions without much occlusion, the locality constraint will tend to keep the algorithm from hypothesizing regions with bad aspect ratios. This happens because a region with a good aspect ratio will tend to survive the smoothing and sampling process until it is small enough to be detected. However, in general, the two constraints do not correlate exactly, particularly when shapes are attached or occlude one another.

6.4. Inter-scale communication and attention

The current implementation for computing Local Rotational Symmetries proceeds from coarse-scale representations to fine-scale representations. For each scale, it does an exhaustive computation of symmetries whose centers lie within a fixed search radius of their boundary points. It also computes wider-scale symmetries for centers and boundary points suggested by regions found at the previous (coarser) scale.

My algorithm for local computation and inter-scale suggestions may be an over-simplification of the algorithm that people use. For example, the algorithm people use might be a more complex iterative process of doing an exhaustive computation, focussing attention on only the most promising locations, and then extending the exhaustive computation to a wider search radius for just those selected locations. For example, it would be easy to produce a version of my algorithm in which the exhaustive search radius varied depending on the density of boundaries in the image. This would have the effect that certain types of regions (e.g. a region with a boundary made up of many disjoint sections of boundary or a region bounded by a thin line) would be salient when they are the only thing in the image, as in Figure 4 (left), but not when there is clutter around them, as in Figure 4 (right).

Context or other features might also be used to direct attention to specific portions of the image or to specific features. For example, regions that might not be noticed in a neutral context may be found if they are specifically pointed out. Sections of boundary with close to constant local curvature might trigger a search at the center predicted by their curvature, beyond the usual search radius. Since the data dependencies in the LRS computation are extremely local, it would not be difficult to produce a version of the computation which could focus attention on suggested locations, e.g. by locally widening parameter settings. I do not know whether any of these things actually happen in human shape perception. Rather, they are examples of factors that it would be worth considering when using detailed psychological evidence to refine this theory of shape representation.

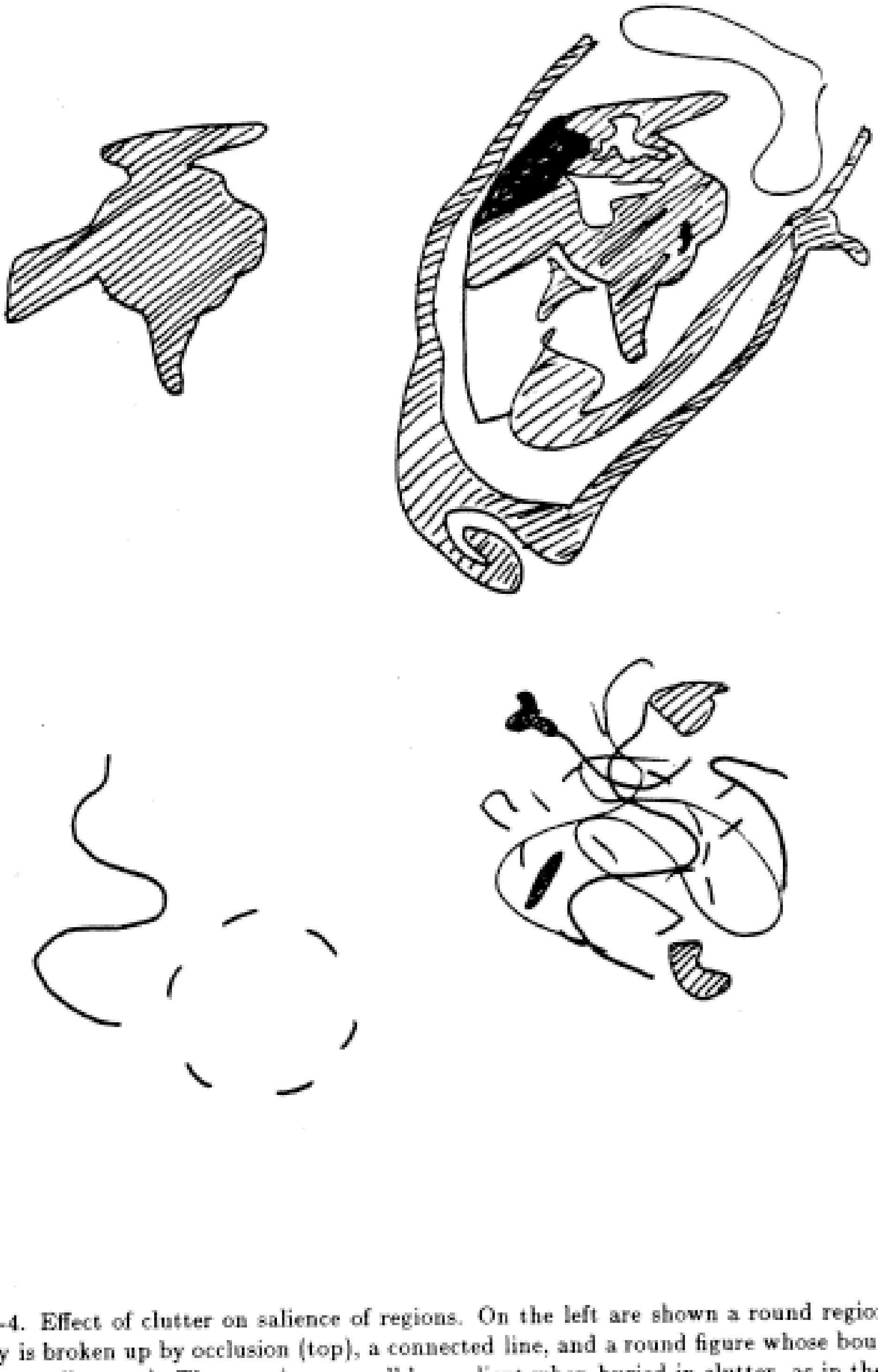

Figure 6-4. Effect of clutter on salience of regions. On the left are shown a round region whose boundary is broken up by occlusion (top), a connected line, and a round figure whose boundary is a dashed line (bottom). These regions are all less salient when buried in clutter, as in the figures on the right.

6.5. Summary

This chapter has described a number of principles involved in computing shape representations at multiple scales of detail. The key ideas involved are:

- Representations should be able to change qualitatively between scales;
- The best of the available techniques for producing multiple-scale representations of visual input is smoothing the grey-scale image, because it is more robust than boundary smoothing and because it produces qualitative changes between scales;
- Computation should be kept local, both for efficiency and because non-local relationships are not perceptually salient;
- Representations at different scales should be related and summarized in a type of analysis similar to Witkin's scale-space analysis of 1-dimensional signals.

These ideas, with the exception of a full scale-space type analysis, were used in the current implementation of the algorithm for computing Local Rotational Symmetry regions. Similar techniques should be also be used in computing Smoothed Local Symmetries. However, the only change that was easy to make, without extensive re-implementation, was to use image smoothing in place of boundary smoothing.

Chapter 7: Conclusion and Future work

In this thesis, I have developed a representation for round regions, Local Rotational Symmetries, that can serve as a companion to the Smoothed Local Symmetry representation for elongated regions. An algorithm for computing these representations has been implemented which computes perceptually reasonable regions from unretouched images of real objects. The high points of this representation include:

- Local Rotational Symmetries are a robust, perceptually reasonable representation for round regions. Smoothed Local Symmetries are unstable on such regions and do not assign them perceptually reasonable analyses.
- These two types of local symmetry representations (1) represent a wider class of shapes than competing representations, at lower computational cost, (2) incorporate a constructive definition of axis or center of a region, and (3) represent and relate both regions and their boundaries.
- Shapes intermediate between two types of shape analysis can have multiple analyses, allowing the shape representation as a whole to be stable.
- The local multiple-scale computation with information passed between scales allows (1) efficient computation, (2) detection of arbitrarily large features, (3) highly detailed and accurate results, (4) representations that are stable under small changes in the input, (5) use of suggestions from coarse scales to focus fine-scale processing.
- The locality restriction causes the representation to avoid generating a class of symmetries that are not perceptually salient.
- The use of grey-scale image smoothing rather than boundary smoothing allows qualitative changes in representation of the sort required for practical reasoning and makes the algorithm work robustly in the presence of gaps and other defects in region boundaries.
- The implementation works robustly on unretouched, natural input images. It does not impose special restrictions such as requiring closed boundaries.

In the previous chapters, I have mentioned a number of areas in which future work could extend or refine the theories presented here:

- Using detailed psychological and psychophysical evidence about human perceptions of shape to refine evaluation metrics, parameters, and algorithms for finding optimal regions;
- Using mathematical techniques commonly used in low-level vision, such as regularization, to find a better algorithm for the optimization problem;
- Finding a version of the non-overlap constraint and non-maximum suppression that is robust and agrees with human perceptions;
- Developing a representation for straight lines and the 2-dimensional half-open regions they bound;
- Developing a region color constraint that is robust and agrees with human perceptions;
- Extending the representations and algorithms to 3-dimensional objects;
- Developing a system for building symbolic representations for round regions from the LRS output;
- Developing a system for building representations for complex regions from the LRS and SLS output;
- Finding algorithms for arbitrating between alternative representations for regions, using both SLS and LRS analyses;
- Building a system to match regions between scales to produce a scale-space analysis of regions in a 2-D image;
- Finding methods for allowing selected long thin regions to be smoothed as if they were the boundaries of grey-scale regions.

Another topic for further research is to build a new implementation of the code for computing Smoothed Local Symmetries, incorporating the following ideas from my LRS implementation:

- Detecting the infinite degeneracies on round regions and lines in a principled manner;
- Smoothing the grey-scale image, rather than the boundaries;
- Matching regions from adjacent scales so that information can be passed between scales;
- Using a local multi-scale algorithm to compute exact symmetries;

- Perhaps allowing inexact symmetries, particularly when searching at a fine scale for a region found at a coarser scale.

In fact, the Smoothed Local Symmetry code used to produce examples for this thesis has already been altered so that it produces representations at multiple scales by smoothing the grey-scale image, rather than by smoothing the boundaries. This allows the code to run robustly on the region boundaries found in real images, without requiring that the bounding contour of an object be extracted first.

Bibliography

Agin, Gerald Jacob (1972) "Representation and Description of Curved Objects," Stanford Artificial Intelligence Project, AIM-173, STAN-CS-72-305, Stanford University.

Allen, James F. (1984) "Towards a General Theory of Action and Time," *Artificial Intelligence*, 23, 123-154.

Allen, James F. and Johannes A. Koomen (1983) "Planning using a Temporal World Model," Proceedings of the 8th IJCAI.

Asada, Haruo and J. Michael Brady (1984) "The Curvature Primal Sketch," MIT Artificial Intelligence Laboratory, AIM-758.

Bagley, Steven C. (1985) "Using Models and Axes of Symmetry to Describe Two-Dimensional Polygonal Shapes," MS thesis, MIT Department of Electrical Engineering and Computer Science.

Ballard, Dana H. (1981) "Generalizing the Hough Transform to Detect Arbitrary Shapes" *Pattern Recognition*, 13, 111-122.

Dana H. Ballard and Christopher M. Brown (1982) *Computer Vision*, Prentice-Hall, Englewood Cliffs, New Jersey.

Biederman, Irving (unpubl.) "Recognition-by-Components: A Theory of Human Image Understanding."

Blake, Andrew (1983a) "The Least-Disturbance Principle and Weak Constraints," *Pattern Recognition Letters*, 1, 393–399.

______ (1983b) "Parallel Computation in Low-Level Vision," Ph.D. Dissertation, University of Edinburgh, Department of Computer Science.

Blum, Harry (1973) "Biological Shape and Visual Science (Part I)," *J. Theor. Biol.*, 38, 205-287.

Blum, Harry and Roger N. Nagel (1978) "Shape Description using Weighted Symmetric Axis Features," *Pattern Recognition*, 10, 167-180.

Brady, J. Michael (1983) "Criteria for Representations of Shape," pp. 39–84 in A. Rosenfeld and J. Beck, eds., *Human and Machine Vision*, Academic Press, NY.

Brady, J. Michael and Haruo Asada (1984) "Smoothed Local Symmetries and Their Implementation," *International Journal of Robotics Research*, 3/3, 36-61.

Brooks, Rodney A. (1981) "Symbolic Reasoning among 3-D models and 3-D images," *Artificial Intelligence*, 17, 285–348.

Canny, John F. (1983) "Finding Edges and Lines in Images," MIT Artificial Intelligence Laboratory, TR-720.

Connell, Jonathan H. (1985) "Learning Shape Descriptions: Generating and Generalizing Models of Visual Objects," MS thesis, MIT Department of Electrical Engineering and Computer Science.

Connell, Jonathan H. and J. Michael Brady (1985) "Generating and Generalizing Models of Visual Objects," MIT Artificial Intelligence Laboratory, AIM-823.

Crowley, James L. (1982) "A Representation for Visual Information," Carnegie-Mellon Univ., The Robotics Institute, TR-82-7.

Davis, Larry (1982) "Hierarchical Generalized Hough Transforms and Line-Segment Based Generalized Hough Transforms," *Pattern Recognition*, 15, 277-285.

Dowty, David R. (1979) *Word Meaning and Montague Grammar*, D. Reidel, Dordrecht, Holland.

Forbus, Kenneth D. (1984) "Qualitative Process Theory," MIT Artificial Intelligence Laborary, TR-789.

Grimson, W. Eric L. and Theo Pavlidis (1985) "Discontinuity Dectection for Visual Surface Reconstruction," *Computer Vision, Graphics, and Image Processing*, 30, 316-330.

Heide, Scott S. (1984) "A Hierarchical Representation of Shape from Smoothed Local Symmetries," MS thesis, MIT Department Mechanical Engineering.

Hoffman, Donald D. (1978) "Representing Shapes for Visual Recognition," Ph.D. Dissertation, MIT Department of Psychology.

Hollerbach, John M. (1975) "Hierarchical Shape Description of Objects by Selection and Modification of Prototypes," MIT Artificial Intelligence Laboratory, TR-346.

Labov, William (1973) "The Boundaries of Words and their Meanings," pp. 340-373 in Charles-James N. Bailey and Roger W. Shuy, eds., *New Ways of Analyzing Variation in English*, Georgetown Univ. Press, Washington, D.C.

Marr, David (1982) *Vision*, W.H. Freeman and Co., San Francisco.

Marr, David and H.K. Nishihara (1978) "Representation and Recognition of the Spatial Organization of Three-dimensional Structure" *Proc. Roy. Soc. B.*, Vol. 200, pp. 269–294.

Mayhew, J.E.W. and J.P. Frisby (1981) "Psychophysical and Computational Studies towards a Theory of Human Stereopsis," *Artificial Intelligence*, 17, 349-385.

Pavlidis, Theo (1977) *Structural Pattern Recognition*, Springer-Verlag, Berlin.

______ (1982) *Algorithms for Graphics and Image Processing*, Computer Science Press.

Patil, Ramesh S. (1981) "Causal Representation of Patient Illness for Electolyte and Acid-Base Diagnosis," MIT Laboratory for Computer Science, TR-267.

Petrides, George A. (1972) *A Field Guide to Trees and Shrubs*, second edition, The Peterson Field Guide Series, 11, Houghton Mifflin Co., Boston.

Ponce, Jean and J. Michael Brady (1985) "Toward a Surface Primal Sketch," Takeo Kanade, ed., *Three-Dimensional Visual Systems*, Kluwer Academic Publishers, Hingham, MA.

Sakaue, Katsuhiko and Mikio Takagi (1980) "Separation of Overlapping Particles by Iterative Method," Proceedings of the 5th International Conference of Pattern Recognition, IEEE.

______ (1982) "Image Segmentation by Iterative Method," Proceedings of the 6th International Conference of Pattern Recognition, IEEE.

Shani, Uri and Dana H. Ballard (1984) "Splines as Embeddings for Generalized Cylinders," *Computer Vision, Graphics, and Image Processing*, 27, 129-156.

Tanimoto, Steven L. (1978) "Regular Hierarchical Image and Processing Structures in Machine Vision," pp. 165-174 in Allen R. Hanson and Edward M. Riseman, *Computer Vision Systems*, Academic Press, NY.

Taylor, Barry (1977) "Tense and Continuity," *Linguistics and Philosophy*, 1, 199-220.

Terzopoulos, Demetri (1984) "Multiresolution Computation of Visible-Surface Representations," Ph.D. Dissertation, MIT Department of Electrical Engineering and Computer Science.

Torre, V. and T. Poggio (1984) "On Edge Detection," MIT Artificial Intelligence Laboratory, AIM-768.

Watt, R.J. and M.J. Morgan (to appear) "A Theory of the Primitive Spatial Code in Human Vision," to appear in *Vision Research.*

Weld, Daniel S. (1984) "Switching Between Discrete and Continuous Process Models to Predict Genetic Activity," MIT Artificial Intelligence Laboratory, TR-793.

Witkin, Andrew P. (1983) "Scale-Space Filtering," Proceedings of the 8th IJCAI.

Zusne, Leomard (1970) *Visual Perception of Form*, Academic Press, NY.

Appendix: The Code

The following pages list the ZETALISP code for computing Local Rotational Symmetry regions. Only the code for computing the symmetries and building connected regions is shown. The complete system also contains code for smoothing and sampling images, code for displaying results and partial results, code for saving results in files and and reading them back in, and the code for the edge finder described in Canny (1983).

```
;;------------------------------------------------------------
;; Overview
;;------------------------------------------------------------

;; The input to the code is a list of objects of flavor
;;    lrs-analysis-at-scale.  Initially, each of these
;;    analyses contains only a smoothed version of the original
;;    image and the boundaries found by the edge finder for that image.
;;    The analyses are typically created so that the image is shrunk
;;    by a factor of sqrt(2) in each dimension between adjacent
;;    scales.  The analyses are ordered in the list, with coarser
;;    scales first.

(defflavor lrs-analysis-at-scale
        (expansion-factor ;; how much image was shrunk
         computed-p       ;; have regions been found for this scale yet?
          smoothed-image  ;; smoothed grey-scale image
          orientation-array  ;; boundaries from this image
          all-regions     ;; list of LRS regions for all centers
          best-regions    ;; LRS regions after non-maximum suppression
          suggestion-center-list  ;; center locations suggested by
                                  ;; regions at previous coarser scale
          suggestions)  ;; suggestions from previous (coarser) scale
        ()
  :gettable-instance-variables
  :settable-instance-variables)

;; To compute LRS regions, the function lrs-multi-scale is called
;;     on this list of analysis objects.

;;------------------------------------------------------------
;;  Multi-scale code
;;------------------------------------------------------------

(defmethod (lrs-analysis-at-scale :compute-regions)
        (&optional (window nil) (max-radius 15)
        (max-deviation 500) (max-angle-distance 500)
        (min-evaluation 5.0) (max-average-deviation 200)
        (min-percentage 10) (max-percentage-for-distinctness 50))
  (setq all-regions
        (compute-all-regions-for-image-with-suggestions
          orientation-array suggestion-center-list suggestions
          window max-radius max-deviation max-angle-distance
          min-evaluation max-average-deviation min-percentage
          max-percentage-for-distinctness))
  (format t "~%Picking best regions")
  (setq best-regions
        (get-local-best-regions
           all-regions max-percentage-for-distinctness))
  (setq computed-p t))
```

```
;; Called for side-effect.
;; This function computes LRS regions at multiple scales, passing
;;   down suggestions from coarser to finer scales.
;; When the best LRS regions are computed for a given scale, they
;;   are displayed on <window>.
(defun lrs-multi-scale (converted-meltdown
                         &optional (window nil) (scale 1.0)
                         (max-radius 8)
                         (max-deviation 500) (max-angle-distance 500)
                         (min-evaluation 7.0) (max-average-deviation 200)
                         (min-percentage 10)
                         (max-percentage-for-distinctness 50)
                         (expansion-factor (sqrt 2)))
  (do ((mylist converted-meltdown (cdr mylist)))
      ((null mylist))
    (format t "~%~%Computing regions for scale ~a"
            (send (car mylist) ':expansion-factor))
    (send (car mylist) ':compute-regions
           nil ;;window
          max-radius max-deviation max-angle-distance
          min-evaluation max-average-deviation
          min-percentage max-percentage-for-distinctness)
    (format t "~%~a good regions found"
            (length (send (car mylist) ':best-regions)))
    (dolist (region (send (car mylist) ':best-regions))
      (format t "  ~a" (car region)))
    (cond ((and window (send (car mylist) ':best-regions))
           (send window ':nrefresh)
           (send window ':set-cursorpos 50 50)
           (send window ':string-out
                 (format nil "The best regions for scale ~a."
                                      (send (car mylist) ':expansion-factor)))
           (send window ':set-cursorpos 0 0)
           (send (car mylist)
                 ':dot-display-orientation-array
                 window 100 100 scale)
           (dolist (region (send (car mylist) ':best-regions))
             (display-center-boundary (nth 3 region)
                                      (nth 1 region)
                                      (nth 2 region)
                                      window 100 100
                                      (* scale
                                         (send (car mylist)
                                               ':expansion-factor))))))
    (cond ((not (null (cdr mylist)))
           (pass-down-suggestions (car mylist) (cadr mylist)
                                  expansion-factor)))))
```

```
(defun pass-down-suggestions (current-analysis next-analysis expansion-factor)
  (format t "~%Adding ~a suggestions."
      (length (send current-analysis ':best-regions)))
  (let ((count 0))
    (setq count 0)
    (dolist (region (send current-analysis ':best-regions))
      (setq count (1+ count))
      (format t "  ~a" count)
      (send next-analysis
            ':set-suggestions
            (cons
              (make-suggestion-from-region
                region
                (// (send current-analysis ':expansion-factor)
                    (send next-analysis ':expansion-factor))
                expansion-factor)
              (send next-analysis ':suggestions)))))
  (dolist (suggestion (send next-analysis ':suggestions))
    (dolist (center (car suggestion))
      (cond ((not (member center
                          (send next-analysis ':suggestion-center-list)))
             (send next-analysis ':set-suggestion-center-list
                   (cons center
                         (send next-analysis ':suggestion-center-list))))))))



;;-------------------------------------------------------------
;; Computation of radii and orientations
;;-------------------------------------------------------------

;; For each boundary-point location in mask, mask contains orientation of
;;    point from mask center.
(defun make-center-orientation-mask (max-radius)
  (let ((mask (make-array (list (1+ (* 2 max-radius))
                                (1+ (* 2 max-radius)))
                          ':type 'art-16b))
        (size (1+ (* 2 max-radius)))
        (delta-x nil)
        (delta-y nil)
        (orientation nil))
    (dotimes (y size)
      (setq delta-y (- y max-radius))
      (dotimes (x size)
        (setq delta-x (- x max-radius))
        (cond ((and (= 0 delta-x)
                    (= 0 delta-y)))
              (t
               (setq orientation (ceiling (// (* 1800 (atan delta-y delta-x))
                                              pi)))
               (cond ((<= orientation 0)  ;; i.e., I think, if equal to zero
                      (setq orientation (+ orientation 3600))))
               (aset orientation mask x y)))))
    mask))
```

```
;; For each boundary-point location in mask, mask contains radius from
;;     point to mask center.
(defun make-radius-mask-without-missing-corners (min-radius max-radius)
  (let ((mask (make-array (list (1+ (* 2 max-radius))
                                (1+ (* 2 max-radius)))
                          ':type 'art-16b))
        (size (1+ (* 2 max-radius)))
        (delta-x nil)
        (delta-y nil)
        (radius nil))
    (fillarray mask '(0))
    (dotimes (y size)
      (setq delta-y (- y max-radius))
      (dotimes (x size)
        (setq delta-x (- x max-radius))
        (setq radius (floor (sqrt (+ (* delta-x delta-x)
                                     (* delta-y delta-y)))))
        (cond ((>= radius min-radius)
               (aset radius mask x y)))))
    mask))

(defvar *cm-mask-max-radius* 20)

(defvar *cm-radius-mask*
        (make-radius-mask-without-missing-corners 0 *cm-mask-max-radius*))

(defvar *cm-orientation-mask*
        (make-center-orientation-mask *cm-mask-max-radius*))

;; Uses one mask of size 20 to compute radii.
;; Estimates values for larger displacements based on values from
;;   the small array.
(defun get-estimated-radius (delta-x delta-y)
  (do* ((factor 1 (* 2 factor)))
       ((and (<= (abs (round delta-x factor)) *cm-mask-max-radius*)
             (<= (abs (round delta-y factor)) *cm-mask-max-radius*))
        (* factor (aref *cm-radius-mask*
                        (+ (round delta-x factor) *cm-mask-max-radius*)
                        (+ (round delta-y factor) *cm-mask-max-radius*))))))

;; Uses one mask of size 20 to compute orientations.
;; Estimates values for larger displacements based on values from
;;   the small array.
(defun get-estimated-orientation (delta-x delta-y)
  (do* ((factor 1 (* 2 factor)))
       ((and (<= (abs (round delta-x factor)) *cm-mask-max-radius*)
             (<= (abs (round delta-y factor)) *cm-mask-max-radius*))
        (aref *cm-orientation-mask*
              (+ (round delta-x factor) *cm-mask-max-radius*)
              (+ (round delta-y factor) *cm-mask-max-radius*)))))
```

```
;; The two orientations should be in the range (0,3600]
(defun orientation-distance (orientation-1 orientation-2)
  ;; normalize orientation-1 to a coordinate system in which
  ;;   orientation-2 is zero degrees
  (setq orientation-1 (- orientation-1 orientation-2))
  (cond ((> orientation-1 1800)
         (setq orientation-1 (- orientation-1 3600)))
        ((< orientation-1 -1800)
         (setq orientation-1 (+ orientation-1 3600))))
  ;; get the distance between orientation-1 and either 0 or 1800,
  ;;   depending on which is closer
  (min (abs orientation-1)
       (- 1800 (abs orientation-1))))

;;-----------------------------------------------------------------
;; Box for storing points as displacements from center
;;-----------------------------------------------------------------

;; The idea behind these functions is to be able to store points
;;   at any displacement and be able to give at least an estimate
;;   of their orientation and radius, as fast as possible.
;;   Since a few computations create points with large displacements,
;;   but most computation is done with points closer in, these
;;   structures provide fast access to the near points while still
;;   allowing one to add points further away.

(defstruct (center-map :named
                       (:print "[map of points from center (~a,~a)]"
                        (center-map-center-x center-map)
                        (center-map-center-y center-map))
                       :conc-name)
  center-x
  center-y
  full-map
  full-map-radius
  sparse-list)  ;; an assoc list

(defun create-center-map (max-radius center-x center-y)
  (make-center-map
    center-x center-x
    center-y center-y
    full-map (make-array (list (1+ (* 2 max-radius))
                               (1+ (* 2 max-radius)))
                         ':type 'art-16b)
    full-map-radius max-radius
    sparse-list nil))

(defun map-max-radius (center-map)
  (let ((current-max-radius (center-map-full-map-radius center-map)))
    (dolist (point (center-map-sparse-list center-map))
      (setq current-max-radius
            (max current-max-radius
                 (car (car point))
                 (cadr (car point)))))
    current-max-radius))
```

```
(defun copy-center-map (old-map new-map-radius)
  (let ((new-map (create-center-map new-map-radius
                                    (center-map-center-x old-map)
                                    (center-map-center-y old-map)))
        (old-map-radius (center-map-full-map-radius old-map))
        (temp-value))
    (do ((delta-x (- old-map-radius) (1+ delta-x)))
        ((> delta-x old-map-radius))
      (do ((delta-y (- old-map-radius) (1+ delta-y)))
          ((> delta-y old-map-radius))
        (setq temp-value (get-point delta-x delta-y old-map))
        (cond ((> temp-value 0)
               (add-point temp-value delta-x delta-y new-map)))))
    (dolist (point (center-map-sparse-list old-map))
      (add-point (cadr point)
                 (car (car point))
                 (cadr (car point))
                 new-map))
    new-map))

;; Deviations are always made >0 so that 0 can indicate
;;   the absence of a boundary point.
(defun add-point (deviation delta-x delta-y center-map)
  (cond ((= 0 deviation)
         (setq deviation 1)))
  (cond ((and (<= (abs delta-x)
                  (center-map-full-map-radius center-map))
              (<= (abs delta-y)
                  (center-map-full-map-radius center-map)))
         ;; i.e. it is in the area of the full map
         (aset deviation
               (center-map-full-map center-map)
               (+ delta-x (center-map-full-map-radius center-map))
               (+ delta-y (center-map-full-map-radius center-map))))
        (t
         (alter-center-map center-map
                           sparse-list
                           (cons
                             (list (list delta-x delta-y)
                                   deviation)
                             (center-map-sparse-list center-map)))))
  nil)
```

```
;; returns deviation, or 0 if no point at that location
(defun get-point (delta-x delta-y center-map)
  (cond ((and (<= (abs delta-x)
                  (center-map-full-map-radius center-map))
              (<= (abs delta-y)
                  (center-map-full-map-radius center-map)))
         ;; i.e. it is in the area of the full map
         (aref (center-map-full-map center-map)
               (+ delta-x (center-map-full-map-radius center-map))
               (+ delta-y (center-map-full-map-radius center-map))))
        (t  ;; otherwise it is in the assoc list
         (let ((raw-result (assoc (list delta-x delta-y)
                                  (center-map-sparse-list center-map))))
           (cond ((null raw-result)
                  0)
                 (t
                  (cadr raw-result)))))))


;;---------------------------------------------------------------
;; Computing a map of deviations from normal for a center location
;;---------------------------------------------------------------

;; Are there any points in this region?
(defun quick-check (orientation-array center-x center-y max-radius)
  (let ((x-size (car (array-dimensions orientation-array)))
        (y-size (cadr (array-dimensions orientation-array))))
    (do ((real-x (max  0 (- center-x max-radius)) (1+ real-x)))
        ((> real-x (min (+ center-x max-radius) x-size))
         nil)
      (cond ((do ((real-y (max 0 (- center-y max-radius))
                          (1+ real-y)))
                 ((> real-y (min (+ center-y max-radius) y-size))
                  nil)
               (cond ((and (array-in-bounds-p orientation-array real-x real-y)
                           (> (aref orientation-array real-x real-y) 0))
                      (return t))))
             (return t))))))

(defun map-good-centers (orientation-array max-radius)
  (let ((center-map (make-equal-hash-table))
        (x-size (car (array-dimensions orientation-array)))
        (y-size (cadr (array-dimensions orientation-array))))
    (format t "~%")
    (do ((center-x (- max-radius) (1+ center-x)))
        ((> center-x (+ max-radius x-size)))
      (format t " ~a" (- (+ max-radius x-size) center-x))
      (do ((center-y (- max-radius) (1+ center-y)))
          ((> center-y (+ max-radius y-size)))
        (cond ((quick-check orientation-array center-x center-y max-radius)
               (puthash (list center-x center-y)
                        (compute-map-for-center orientation-array
                                                center-x center-y
                                                max-radius)
                        center-map)))))
    center-map))
```

```
;; correction for funny orientation convention in Canny code
(defmacro correct-orientation (orientation-array x y)
  `(let ((raw-orientation (aref ,orientation-array ,x ,y)))
     (cond ((> raw-orientation 1800)
            (- raw-orientation 1800))
           (t (+ raw-orientation 1800)))))


;; Excludes black points in the 8 neighbors of the center.
;; Array returned contains the deviations from normality for boundary
;;   points with deviation below max-deviation.
;; These deviations are always at least 1, so that zeros in the array
;;   can indicate absence of boundary point.
;; If you supply a center-map, it will clear it and use it.
;;   You are responsible for seeing that it was the right radius!
(defun compute-map-for-center (orientation-array center-x center-y max-radius
                               &optional (center-map nil))
  (cond (center-map
         (alter-center-map center-map
                           sparse-list nil
                           center-x center-x
                           center-y center-y)
         (fillarray (center-map-full-map center-map) '(0)))
        (t
         (setq center-map (create-center-map max-radius center-x center-y))))
  (do ((delta-x (- max-radius) (1+ delta-x))
       (real-x (- center-x max-radius) (1+ real-x)))
      ((> delta-x max-radius))
    (do ((delta-y (- max-radius) (1+ delta-y))
         (real-y (- center-y max-radius) (1+ real-y)))
        ((> delta-y max-radius))
      (cond ((and (array-in-bounds-p orientation-array
                                     real-x real-y)
                  (or (> (abs delta-x) 1)
                      (> (abs delta-y) 1)))
             (cond ((> (aref orientation-array real-x real-y) 0)
                    ;; i.e. there is a boundary point at this location
                    (add-point (orientation-distance
                                (get-estimated-orientation delta-x delta-y)
                                (correct-orientation
                                  orientation-array real-x real-y))
                               delta-x delta-y
                               center-map)))))))
  center-map)
```

```
;;----------------------------------------------------------------
;; Get regions for image
;;----------------------------------------------------------------

(defun compute-all-regions-for-image-with-suggestions
       (orientation-array suggestion-centers suggestions
        &optional (window nil) (max-radius 15)
        (max-deviation 500) (max-angle-distance 500)
        (min-evaluation 5.0)(max-average-deviation 200)
        (min-percentage 10) (max-percentage-for-distinctness 50))
  (let ((resultlist nil)
        (center-map (create-center-map max-radius 0 0))
        (x-size (car (array-dimensions orientation-array)))
        (y-size (cadr (array-dimensions orientation-array))))
    (format t "~%")
    (do ((center-x (- max-radius) (1+ center-x)))
        ((> center-x (+ max-radius x-size)))
      (format t " ~a" (- (+ max-radius x-size) center-x))
      (do ((center-y (- max-radius) (1+ center-y)))
          ((> center-y (+ max-radius y-size)))
        (setq resultlist
              (append (compute-regions-for-center-with-suggestions
                         orientation-array suggestions center-x center-y window
                         center-map
                         max-radius max-deviation max-angle-distance
                         min-evaluation max-average-deviation  min-percentage
                         max-percentage-for-distinctness)
                       resultlist))))
    (format t "~%and ~a centers provided by suggestions:  "
            (length suggestion-centers))
      (dolist (center-point suggestion-centers)
        (cond ((and (or (< (car center-point) (- max-radius))
                        (> (car center-point) (+ max-radius x-size)))
                    (or (< (cadr center-point) (- max-radius))
                        (> (cadr center-point) (+ max-radius y-size))))
               (format t ".")
               ;; i.e. center-point in suggestion wasn't in exhaustive search
               (setq resultlist
                     (append (compute-regions-for-center-with-suggestions
                                orientation-array suggestions
                                (car center-point) (cadr center-point) window
                                center-map
                                max-radius max-deviation max-angle-distance
                                min-evaluation max-average-deviation
                                min-percentage
                                max-percentage-for-distinctness)
                              resultlist)))))
    resultlist))

 ;; each region is of the form (evaluation center-x center-y boundary)
 (defun get-best (regionlist)
   (let ((current-best (car regionlist)))
    (dolist (region regionlist)
       (cond ((> (car region) (car current-best))
              (setq current-best region))))
    current-best))
```

```
(defun get-local-best-regions (regionlist max-percentage-distinct)
  (let ((resultlist nil))
    (do ((current-best (get-best regionlist) (get-best myregionlist))
         (myregionlist regionlist)
         (templist nil nil))
        ((null current-best) resultlist)
      (push current-best resultlist)
      (dolist (region myregionlist)
        (cond ((not (boundary-similar (nth 3 region) (nth 3 current-best)
                                 (- (nth 1 region) (nth 1 current-best))
                                 (- (nth 2 region) (nth 2 current-best))
                                 max-percentage-distinct))
               (push region templist))))
      (setq myregionlist templist)
      (format t "  ~a" (length myregionlist))
      (cond ((null myregionlist)
             (return resultlist))))))

;;-----------------------------------------------------------------
;; Get possible regions for center
;;-----------------------------------------------------------------

;; suggestions is a list of items of the form
;; <list of center points, list of boundary points>
(defun add-suggestions
  (center-map center-x center-y suggestions orientation-array)
  (dolist (suggestion suggestions)
    (cond ((member (list center-x center-y) (car suggestion))
           (dolist (boundary-point (cadr suggestion))
             (cond ((and (array-in-bounds-p orientation-array
                                 (+ center-x (car boundary-point))
                                 (+ center-y (cadr boundary-point)))
                         (or (> (abs (car boundary-point)) 1)
                             (> (abs (car boundary-point)) 1)))
                    (cond ((> (aref orientation-array
                                    (+ center-x (car boundary-point))
                                    (+ center-y (cadr boundary-point)))
                              0)
                           (add-point
                             (orientation-distance
                               (get-estimated-orientation
                                 (car boundary-point)
                                 (cadr boundary-point))
                               (correct-orientation
                                  orientation-array
                                  (+ center-x (car boundary-point))
                                  (+ center-y (cadr boundary-point))))
                             (car boundary-point)
                             (cadr boundary-point)
                             center-map)))))))))
  center-map)
```

```
;; Returns a list of regions
(defun compute-regions-for-center-with-suggestions
       (orientation-array suggestions center-x center-y
        &optional (window nil) (center-map nil) (max-radius 15)
        (max-deviation 500) (max-angle-distance 500)
        (min-evaluation 5.0) (max-average-deviation 200)
        (min-percentage 10) (max-percentage-for-distinctness 50))
  (let ((suggestion-p nil)
        (resultlist nil)
        (temp-map))
    (dolist (suggestion suggestions)
      (cond ((member (list center-x center-y) (car suggestion))
             (setq suggestion-p t))))
    (cond ((or (quick-check orientation-array center-x center-y max-radius)
               suggestion-p)
           (setq temp-map
                 (add-suggestions
                   (compute-map-for-center orientation-array
                                           center-x center-y
                                           max-radius center-map)
                   center-x center-y
                   suggestions orientation-array))
           ;; expand out if necessary
           (cond ((> (length (center-map-sparse-list temp-map))
                     30)
                  (format t "#")
                           ;;"(new map radius ~a)" (map-max-radius temp-map))
                  (setq temp-map (copy-center-map temp-map
                                            (map-max-radius temp-map)))))
           (setq resultlist
                 (get-local-maxima
                   (make-all-regions
                     temp-map
                     max-deviation max-angle-distance
                     min-evaluation max-average-deviation min-percentage)
                   max-percentage-for-distinctness))))
    (cond ((and resultlist window)
           (send window ':nrefresh)
           (dolist (region resultlist)
             (display-center-boundary
             (nth 3 region) (nth 1 region) (nth 2 region)
                                 window 100 100 1.0))))
    resultlist))


(defun point-is-in-boundary (point-x point-y boundary)
  (do ((mylist boundary (cdr mylist)))
      ((null mylist) nil)
    (cond ((and (= point-x (nth 2 (car mylist)))
                (= point-y (nth 3 (car mylist))))
           (return t)))))
```

```
;; NOT SYMMETRIC in the two boundaries
;; delta-center-x is the distance from better-boundary's center to
;;      new-boundary's center, and similarly for delta-center-y
(defun boundary-similar
    (new-boundary better-boundary delta-center-x delta-center-y
                        &optional (max-percentage 50))
  (let ((summ 0))
    (dolist (point new-boundary)
      (cond ((point-is-in-boundary (+ (nth 2 point) delta-center-x)
                                   (+ (nth 3 point) delta-center-y)
                                   better-boundary)
             (setq summ (1+ summ)))))
    (cond ((> (ceiling (* summ 100)
                      (length new-boundary)) ;; percentage same points
              max-percentage)
           T)
          (t NIL))))

;; returns packages of <evaluation center-x center-y boundary>
(defun make-all-regions (center-map
                         &optional
                         (max-deviation 500)
                         (max-angle-distance 500)
                         (min-evaluation 5.0)
                         (max-average-deviation 200)
                         (min-percentage 10))
  (let ((resultlist nil)
        (current-evaluation nil))
    (do ((current-max-deviation 200 (+ 50 current-max-deviation)))
        ((> current-max-deviation max-deviation))
      (dolist (boundary
                (make-all-joins
                  (gather-all-curves center-map current-max-deviation)
                  current-max-deviation max-angle-distance))
        (setq current-evaluation
              (get-evaluation-of-boundary
                boundary min-percentage max-average-deviation))
        (cond ((>= current-evaluation
                   min-evaluation)
               (push (list current-evaluation
                           (center-map-center-x center-map)
                           (center-map-center-y center-map)
                           boundary)
                     resultlist)))))
    resultlist))

;; same center!!!!
(defun any-boundary-similar (new-boundary better-boundarylist max-percentage)
  (do ((mylist better-boundarylist (cdr mylist)))
      ((null mylist) nil)
    (cond ((boundary-similar (nth 3 new-boundary)
                             (nth 3 (car mylist))
                             0 0 max-percentage)
           (return T)))))
```

```
;; all boundaries from same center
;; each boundary is a package <evaluation center-x center-y boundary>
(defun get-local-maxima (boundarylist max-percentage)
  (let ((newlist
          (sort (copylist boundarylist)
                (function (lambda (boundary1 boundary2)
                            (> (car boundary1) (car boundary2))))))
        (returnlist nil))
    (dolist (boundary newlist)
      (cond ((not (any-boundary-similar boundary returnlist max-percentage))
             (push boundary returnlist))))
    returnlist))

;;---------------------------------------------------------------
;; Gather connected curves for center
;;---------------------------------------------------------------

;; Returns a curve going counter-clockwise containing the starting point
;;   Halts curve at 3-way joins and gaps. Expects point given it to be black.
;;   Leaves a 1 in temp-array at each of the points in the curve.
;; Returns the new boundary and the altered version of temp-list
(defun get-connected-curve (center-map start-x start-y max-deviation temp-map)
  (cond ((= 0 (get-point  start-x start-y center-map))
         (ferror nil
            "~%Get connected curve called with non-black starting point.")))
  (let ((resultlist (list (list 'real (get-point start-x start-y center-map)
                               start-x start-y))))
    (add-point 1 start-x start-y temp-map)
    (do ((next-points (get-neighbors-positive center-map start-x start-y
                                              max-deviation temp-map)
                      (get-neighbors-positive center-map next-x next-y
                                              max-deviation temp-map))
         (next-x nil) (next-y nil))
        ((not (= (length next-points) 1)))
      (setq next-x (caar next-points))
      (setq next-y (cadar next-points))
      (add-point 1 next-x next-y temp-map)
      (push (cons 'real (cons (get-point next-x next-y center-map)
                              (car next-points)))
            resultlist))
    (setq resultlist (nreverse resultlist))
    (do ((next-points (get-neighbors-negative center-map start-x start-y
                                              max-deviation temp-map)
                      (get-neighbors-negative center-map next-x next-y
                                              max-deviation temp-map))
         (next-x nil)
         (next-y nil))
        ((not (= (length next-points) 1)))
      (setq next-x (caar next-points))
      (setq next-y (cadar next-points))
      (add-point 1 next-x next-y temp-map)
      (push (cons 'real (cons (get-point next-x next-y center-map)
                              (car next-points)))
            resultlist))
    resultlist))
```

```
(defmacro new-point-there-p (center-map x y max-deviation temp-map)
  '(and (= 0 (get-point ,x ,y ,temp-map))
        (> (get-point ,x ,y ,center-map) 0)
        (<= (get-point ,x ,y ,center-map) ,max-deviation)))

;; returns a list of lists, each representing a boundary
(defun gather-all-curves (center-map max-deviation)
  (let ((temp-map (create-center-map
                   (center-map-full-map-radius center-map)
                   (center-map-center-x center-map)
                   (center-map-center-y center-map)))
        (radius (center-map-full-map-radius center-map))
        (returnlist nil)
        (new-boundary nil))
    ;; all the points in the full array of the map
    (do ((x (- radius) (1+ x)))
        ((> x radius))
      (do ((y (- radius) (1+ y)))
          ((> y radius))
        (cond ((new-point-there-p center-map x y max-deviation temp-map)
               (setq new-boundary
                     (get-connected-curve
                        center-map x y max-deviation temp-map))
               (push new-boundary returnlist)))))
    (dolist (point (center-map-sparse-list center-map))
      (cond ((new-point-there-p center-map
                                (caar point) (cadar point)
                                max-deviation temp-map)
             (setq new-boundary
                   (get-connected-curve center-map (caar point) (cadar point)
                                        max-deviation temp-map))
             (push new-boundary returnlist))))
    returnlist))

;; Going counter-clockwise, i.e. clockwise on display
(defun get-neighbors-positive (center-map x y max-deviation temp-list)
  (let ((returnlist nil))
   (do ((new-x (1- x) (1+ new-x)))
       ((> new-x (1+ x)))
     (do ((new-y (1- y) (1+ new-y)))
         ((> new-y (1+ y)))
       (cond ((and (not (and (= x new-x) (= y new-y)))
                   (new-point-there-p
                     center-map new-x new-y max-deviation temp-list)
                   (> (- (* new-y x) (* new-x y)) 0))
              (push (list new-x new-y) returnlist)))))
   returnlist))
```

```
;; Going clockwise, i.e. counter-clockwise on display
(defun get-neighbors-negative (center-map x y max-deviation temp-list)
  (let ((returnlist nil))
   (do ((new-x (1- x) (1+ new-x)))
        ((> new-x (1+ x)))
      (do ((new-y (1- y) (1+ new-y)))
          ((> new-y (1+ y)))
        (cond ((and (not (and (= x new-x) (= y new-y)))
                    (new-point-there-p
                      center-map new-x new-y max-deviation temp-list)
                    (< (- (* new-y x) (* new-x y)) 0))
               (push (list new-x new-y) returnlist)))))
    returnlist))

;;-----------------------------------------------------------------
;;  Describing and evaluating a region
;;-----------------------------------------------------------------

(defun new-boundary-closed-p (boundary)
  (and (> (length boundary) 2)
       (<= (abs (- (nth 2 (car boundary))
                   (nth 2 (car (last boundary)))))
           1)
       (<= (abs (- (nth 3 (car boundary))
                   (nth 3 (car (last boundary)))))
           1)))

(defun get-average-deviation (boundary)
  (cond ((null boundary)
         (ferror nil "~%Can't evaluate null boundary."))
        (t (let ((total-deviation 0)
                 (number-of-points 0))
             (dolist (point boundary)
               (setq total-deviation (+ total-deviation (nth 1 point)))
               (setq number-of-points (1+ number-of-points)))
             (ceiling total-deviation number-of-points)))))

(defun get-percent-real (boundary)
  (cond ((null boundary)
         (ferror nil "~%Can't evaluate null boundary."))
        (t
         (let ((total-real-points 0)
               (total-points 0))
           (dolist (point boundary)
             (setq total-points (1+ total-points))
             (cond ((eq (nth 0 point) 'REAL)
                    (setq total-real-points (1+ total-real-points)))))
           (ceiling (* 100 total-real-points) total-points)))))
```

```
(defun get-angle-percentage (boundary)
  (cond ((null boundary) 0)
        ((new-boundary-closed-p boundary) 100)
        (t (let* ((start-orientation
                    (get-estimated-orientation (nth 2 (car boundary))
                                                (nth 3 (car boundary))))
                  (current-orientation-difference nil)
                  (full-loop-count 0)
                  (past-180-degree-flag nil))
             (dolist (point boundary)
               (setq current-orientation-difference
                     (- (get-estimated-orientation (nth 2 point) (nth 3 point))
                        start-orientation))
               (cond ((< current-orientation-difference 0)
                      (setq current-orientation-difference
                            (+ current-orientation-difference 3600))))
               (cond ((and past-180-degree-flag
                           (< current-orientation-difference 1800))
                      (setq past-180-degree-flag nil)
                      (setq full-loop-count (1+ full-loop-count)))
                     ((and (>= current-orientation-difference 3300)
                           (not past-180-degree-flag))
                      ;; this case happens on adjacent points
                      ;;  if estimated orientation is off so as to screw up
                      ;;  the relative order of their orientations (it can!)
                      (setq current-orientation-difference 0))
                     ((>= current-orientation-difference 1800)
                      (setq past-180-degree-flag t))))
               (+ (* full-loop-count 100) ;; length of full loops around center
                  (ceiling
                   current-orientation-difference 36))))))
                          ;; length of last partial loop

(defun get-evaluation-of-boundary (boundary min-percentage
                                    max-deviation)
  (let ((deviation (get-average-deviation boundary))
        (angle-percentage (get-angle-percentage boundary)))
    (cond ((or (< angle-percentage min-percentage)
               (> deviation max-deviation))
           0)
          ((new-boundary-closed-p boundary)
           (+
             (* (min angle-percentage 100)
                (sqrt (// 1.0 deviation)))
             (ceiling (get-percent-real boundary) 50)))
          (t
           (+
             (* (min angle-percentage 90)
                (sqrt (// 1.0 deviation)))
             (ceiling (get-percent-real boundary) 50))))))
```

```
;;-------------------------------------------------------------
;;  Making joins for a center
;;-------------------------------------------------------------

(defun make-all-joins (boundarylist max-deviation max-angle-distance)
  (let ((boundarylist (sort (copylist boundarylist)
                            (function (lambda (boundary1 boundary2)
                                        (> (length boundary1)
                                           (length boundary2))))))
        (returnlist nil))
    (cond ((null boundarylist)
           nil)
          (t
           (do ((boundary (car boundarylist))
                (boundaries boundarylist)
                (added-piece nil nil))
               ((null boundaries)
                (push boundary returnlist))
             (setq added-piece (best-positive-extension
                                boundary boundaries
                                max-deviation max-angle-distance))
             (cond ((eq added-piece boundary)
                    ;; i.e. best extension was to close boundary
                    (setq boundaries (remove added-piece boundaries))
                    (push (close-boundary boundary) returnlist)
                    (setq boundary (car boundarylist)))
                   (added-piece
                    (setq boundaries
                       (remove boundary (remove added-piece boundaries)))
                    (setq boundary (join-boundaries boundary added-piece))
                    (push boundary boundaries))
                       ;; order of execution important here!
                   (t
                    (setq added-piece (best-negative-extension
                                       boundary boundaries
                                       max-deviation max-angle-distance))
                    (cond ((eq added-piece boundary)
                           (ferror nil
                           "~%This case should never happen, because
                                of symmetry"))
                          (added-piece
                           (setq boundaries (remove boundary
                              (remove added-piece boundaries)))
                           (setq boundary
                               (join-boundaries added-piece boundary))
                           (push boundary boundaries))
                          (t
                           (push boundary returnlist)
                           (setq boundaries (remove boundary boundaries))
                           (cond (boundaries
                                  (setq boundary (car boundaries))))))))
           returnlist))))
```

```
;; tries to connect the boundaries in given order
;; if connection is ok, return length of connection
;;    else return NIL
(defmacro connection-okp (boundary1 boundary2 max-angle-distance max-deviation)
  `(let* ((start-point (car (last ,boundary1)))
          (end-point (car ,boundary2))
          (shortest-distance-is-clockwise-p
            (> (- (* (nth 3 end-point) (nth 2 start-point))
                  (* (nth 3 start-point) (nth 2 end-point)))
               0))  ;; get-estimated-orientation can get relative orientations
                    ;;  wrong for points close together.   This condition
                    ;;  catches potential lossage due to this.
          (angle-distance
            (- (get-estimated-orientation
                 (nth 2 end-point) (nth 3 end-point))
               (get-estimated-orientation
                 (nth 2 start-point) (nth 3 start-point))))
          (length-of-connection (length-of-connection
                                  (nth 2 start-point) (nth 3 start-point)
                                  (nth 2 end-point) (nth 3 end-point))))
     (cond ((< angle-distance 0)
            (setq angle-distance (+ angle-distance 3600))))
     (cond ((and shortest-distance-is-clockwise-p
                 (< angle-distance ,max-angle-distance)
                 (< (deviation-of-connection
                      (nth 2 start-point) (nth 3 start-point)
                      (nth 2 end-point) (nth 3 end-point))
                    ,max-deviation)
                 (< length-of-connection
                    (min (max (length ,boundary1) (length ,boundary2))
                         (* 3 (min (length ,boundary1) (length ,boundary2))))))
            length-of-connection)
           (t nil))))

;; returns the boundary from boundarylist that is the best
;;   counter-clockwise extension of boundary, if any are
;;   acceptable extensions
(defun best-positive-extension (boundary boundarylist max-deviation
                                max-angle-distance)
  (let ((best-boundary nil)
        (best-length 77777)
        (current-length nil))
  (dolist (added-boundary boundarylist)
    (setq current-length
          (connection-okp
            boundary added-boundary max-angle-distance max-deviation))
    (cond ((and current-length
                (< current-length best-length))
           (setq best-boundary added-boundary)
           (setq best-length current-length))))
  best-boundary))
```

```
;; returns the boundary from boundarylist that is the best
;;   clockwise extension of boundary, if any are
;;   acceptable extensions
(defun best-negative-extension (boundary boundarylist max-deviation
                                 max-angle-distance)
  (let ((best-boundary nil)
        (best-length 77777)
        (current-length nil))
  (dolist (added-boundary boundarylist)
    (setq current-length
          (connection-okp
             added-boundary boundary max-angle-distance max-deviation))
    (cond ((and current-length
                (< current-length best-length))
           (setq best-boundary added-boundary)
           (setq best-length current-length))))
  best-boundary))


;;------------------------------------------------------------
;;  Joining and displaying boundaries
;;------------------------------------------------------------

(defun display-center-boundary (boundary center-x center-y
                                 window
                                 &optional
                                 (reference-x 0) (reference-y 0)
                                 (scale 1.0))
  (new-draw-x window center-x center-y reference-x reference-y scale)
  (new-draw-cross window center-x center-y reference-x reference-y scale)
  (dolist (point boundary)
    (cond ((eq 'REAL (nth 0 point))
           (send window ':draw-filled-in-circle
                 (normalize-a-point (+ (nth 2 point) center-x)
                                    reference-x scale)
                 (normalize-a-point (+ (nth 3 point) center-y)
                                    reference-y scale)
                 (ceiling scale)))
          (t
           (send window ':draw-circle
                 (normalize-a-point (+ (nth 2 point) center-x)
                                    reference-x scale)
                 (normalize-a-point (+ (nth 3 point) center-y)
                                    reference-y scale)
                 (ceiling scale))))))

;; Presumed to be both going counter-clockwise
(defun join-boundaries (boundary1 boundary2)
  (append boundary1
          (connect-by-line(nth 2 (car (last boundary1)))
                          (nth 3 (car (last boundary1)))
                          (nth 2 (car boundary2))
                          (nth 3 (car boundary2)))
          boundary2))
```

```
(defun close-boundary (boundary)
  (append boundary
          (connect-by-line (nth 2 (car (last boundary)))
                           (nth 3 (car (last boundary)))
                           (nth 2 (car boundary))
                           (nth 3 (car boundary)))))

;; Inward/outward distinction not respected.
(defun orientation-of-points (start-x start-y end-x end-y)
  (cond ((= end-x start-x)
         0)
        (t
         (let ((orientation (floor (+ 900
                                      (// (+ 1800 (atan (- end-y start-y)
                                                        (- end-x start-x)))
                                          pi)))))
           (cond ((>= orientation 3600)
                  (- orientation 3600))
                 (t
                  orientation))))))

;; a quick estimate, hopefully not too gross
(defun deviation-of-connection (start-x start-y end-x end-y)
  (ceiling
    (+
      (orientation-distance
        ;; orientation of filler line
        (orientation-of-points start-x start-y end-x end-y)
        ;; right orientation for midpoint of filler line
        (get-estimated-orientation
          (floor (+ start-x end-x) 2)
          (floor (+ start-y end-y) 2)))
      (orientation-distance
        ;; orientation of filler line
        (orientation-of-points start-x start-y end-x end-y)
        ;; right orientation for one end of filler line
        (get-estimated-orientation
          start-x start-y)))
    2))

(defun length-of-connection (start-x start-y end-x end-y)
  (ceiling (sqrt (+ (expt (- start-x end-x) 2)
                    (expt (- start-y end-y) 2)))))
```

```
;; returns a list of points from start-x,start-y to end-x,end-y
(defun connect-by-line (start-x start-y end-x end-y)
  (let ((delta-x (abs (- end-x start-x)))
        (delta-y (abs (- end-y start-y)))
        (sign-x (cond ((> end-x start-x) 1)
                      (t -1)))
        (sign-y (cond ((> end-y start-y) 1)
                      (t -1)))
        (orientation-of-filler-points
          (orientation-of-points start-x start-y end-x end-y))
        (resultlist nil))
    (do ((previous-x start-x)
         (previous-y start-y))
        ((and (<= (abs (- previous-x end-x)) 1)
              (<= (abs (- previous-y end-y)) 1)))  ;; i.e. adjacent
      (cond ((= (abs (- previous-x end-x))
                (abs (- previous-y end-y)))
                 (setq previous-x (+ sign-x previous-x))
                 (setq previous-y (+ sign-y previous-y)))
            ((> (abs (- previous-x end-x))  ;; more x points to move
                (abs (- previous-y end-y)))
             (cond ((and (> (abs (- previous-y end-y)) 0)
                         (>= (* delta-x (abs (- previous-y end-y)))
                             (* delta-y (abs (- previous-x end-x)))))
                    (setq previous-x (+ sign-x previous-x))
                    (setq previous-y (+ sign-y previous-y)))
                   (t
                    (setq previous-x (+ sign-x previous-x)))))
            (t ;; more y points to move
             (cond ((and (> (abs (- previous-x end-x)) 0)
                         (>= (* delta-y (abs (- previous-x end-x)))
                             (* delta-x (abs (- previous-y end-y)))))
                    (setq previous-x (+ sign-x previous-x))
                    (setq previous-y (+ sign-y previous-y)))
                   (t
                    (setq previous-y (+ sign-y previous-y))))))
      (push (list 'FAKE
                  (orientation-distance
                    orientation-of-filler-points
                    (get-estimated-orientation previous-x previous-y))
                  previous-x previous-y) resultlist))
    (nreverse resultlist)))
```

```
;;------------------------------------------------------------------
;;  Making suggestions
;;------------------------------------------------------------------

;; This is the right way to shift a center location.
;; The subtraction of number-of-expansions is to compensate for an
;;   apparent movement of images created meltdown.  (This correction
;;   is not needed for delta's.)
;; The +/- 2 factor is some slippage for movement of boundaries in
;;   smoothing.
(defun get-points-at-next-scale-for-point
    (x y number-of-expansions expansion-factor)
  (let ((returnlist nil))
    (do ((new-y (floor (* expansion-factor (- (- y number-of-expansions) 2)))
                (1+ new-y)))
        ((> new-y
            (ceiling (* expansion-factor (+ (- y number-of-expansions) 2)))))
      (do ((new-x (floor (* expansion-factor (- (- x number-of-expansions) 2)))
                  (1+ new-x)))
          ((> new-x
              (ceiling (* expansion-factor (+ (- x number-of-expansions) 2)))))
        (push (list new-x new-y) returnlist)))
    returnlist))

;; Takes a delta-x delta-y pair and returns another
(defun get-delta-at-next-scale-for-delta (delta-x delta-y expansion-factor)
  (let ((returnlist nil))
    (do ((new-y (floor (* expansion-factor (- delta-y 2)))
                (1+ new-y)))
        ((> new-y (ceiling (* expansion-factor (+ delta-y 2)))))
      (do ((new-x (floor (* expansion-factor (- delta-x 2)))
                  (1+ new-x)))
          ((> new-x (ceiling (* expansion-factor (+ delta-x 2)))))
        (push (list new-x new-y) returnlist)))
    returnlist))

(defun get-number-of-expansions (expansion-amount expansion-factor)
  (do ((count 0 (1+ count)))
      ((<= expansion-amount (* 1.0 (sqrt expansion-factor)))
       count)
    (setq expansion-amount (// expansion-amount expansion-factor))))
```

```
;; region is of the form <evaluation center-x center-y boundary>
(defun make-suggestion-from-region (region expansion-amount expansion-factor)
  (let ((new-boundary-points nil))
    (dolist (boundary-point (nth 3 region))
      (setq new-boundary-points
            (append (get-delta-at-next-scale-for-delta
                      (nth 2 boundary-point) ;; x
                      (nth 3 boundary-point) ;; y
                      expansion-amount)
                    new-boundary-points)))
    (list (get-points-at-next-scale-for-point
            (nth 1 region)
            (nth 2 region)
            (get-number-of-expansions expansion-amount expansion-factor)
            expansion-amount)
          new-boundary-points)))
```